

1-1-2007

## Design and implementation of NoC routers and their application to Prdt-based NoC's

Shankar Narayanan Neelakrishnan  
*University of Nevada, Las Vegas*

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

---

### Repository Citation

Neelakrishnan, Shankar Narayanan, "Design and implementation of NoC routers and their application to Prdt-based NoC's" (2007). *UNLV Retrospective Theses & Dissertations*. 2253.  
<http://dx.doi.org/10.25669/f0c3-qr7r>

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

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

DESIGN AND IMPLEMENTATION OF NOC ROUTERS AND THEIR  
APPLICATION TO PRDT-BASED NOC'S

by

Shankar Narayanan Neelakrishnan

Bachelor of Engineering in Electrical and Electronics Engineering  
University of Madras, India  
April 2004

A thesis submitted in partial fulfillment  
of the requirements for the

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

**Graduate College**  
**University of Nevada, Las Vegas**  
**Decemeber 2007**

UMI Number: 1452265

### INFORMATION TO USERS

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

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

**UMI<sup>®</sup>**

---

UMI Microform 1452265

Copyright 2008 by ProQuest LLC.

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

ProQuest LLC  
789 E. Eisenhower Parkway  
PO Box 1346  
Ann Arbor, MI 48106-1346



## Thesis Approval

The Graduate College  
University of Nevada, Las Vegas

November 6, 20 07

The Thesis prepared by

Shankar Neelakrishnan

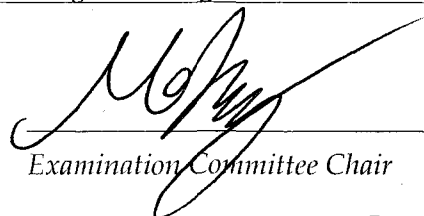

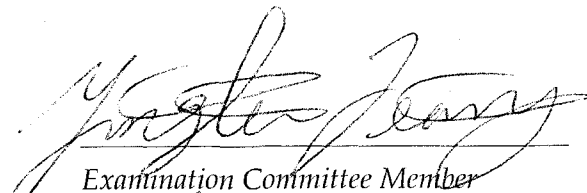
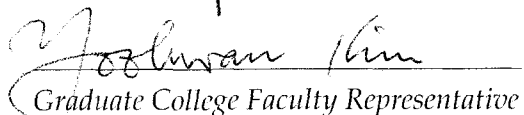
Entitled

"Design and Implementation of NoC Routers and their

Application to PRDT-Based NoC's"

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

Masters of Science in Electrical Engineering

  
Examination Committee Chair  
Dean of the Graduate College  
Examination Committee Member  
Examination Committee Member  
Graduate College Faculty Representative

## ABSTRACT

### **Design and Implementation of NoC Routers and their Application to PRDT-based NoC's**

by

Shankar Narayanan Neelakrishnan

Dr. Mei Yang, Examination Committee Chair  
Assistant Professor of Electrical and Computer Engineering  
University of Nevada, Las Vegas

With a communication-centric design style, Networks-on-Chips (NoCs) emerges as a new paradigm of Systems-on-Chips (SoCs) to overcome the limitations of bus-based communication infrastructure. An important problem in the design of NoCs is the router design, which has great impact on the cost and performance of a NoC system. This thesis is focused on the design and implementation of an optimized parameterized router which can be applied in mesh/torus-based and Perfect Recursive Diagonal Torus (PRDT)-based NoCs.

In specific, the router design includes the design and implementation of two routing algorithms (vector routing and circular coded vector routing), the wormhole switching scheme, the scheduling scheme, buffering strategy, and flow control scheme. Correspondingly, the following components are designed and implemented: input controller, output controller, crossbar switch, and scheduler. Verilog HDL codes are generated and synthesized on ASIC platforms. Most components are designed in parameterized way. Performance evaluation of each component of the router in terms of

timing, area, and power consumption is conducted. The efficiency of the two routing algorithms and tradeoff between computational time ( $t_{\text{setup}}$ ) and area are analyzed.

To reduce the area cost of the router design, the two major components, the crossbar switch and the scheduler, are optimized. Particularly, for crossbar switch, a comparative study of two crossbar designs is performed with the aid of Magic Layout editor, Synopsys CosmosSE and Awaves.

Based on the router design, the PRDT network composed of 4x4 routers is designed and synthesized on ASIC platforms.

## TABLE OF CONTENTS

ABSTRACT.....	iii
LIST OF FIGURES .....	vii
LIST OF TABLES .....	viii
ACKNOWLEDGEMENTS .....	ix
CHAPTER 1 INTRODUCTION .....	1
1.1 Overview of NoC Architecture.....	2
1.1.1 Network Topology .....	3
1.1.2 Routing Algorithm .....	6
1.1.3 Switching Technique .....	8
1.1.4 Buffering Technique .....	10
1.2 Existing NoC Router Design .....	11
1.3 Contribution and Overview of the Thesis .....	13
CHAPTER 2 PRDT AND ITS ROUTING ALGORITHMS .....	15
2.1 Introduction.....	15
2.2 Structure of RDT and PRDT.....	15
2.3 Routing Algorithm for PRDT .....	17
2.3.1 Vector Routing Algorithm .....	18
2.3.2 Circular Coded Vector Routing .....	20
2.3.2.1 Basic functions of shift-code .....	20
2.3.2.2 Intermediate node function .....	21
2.3.2.3 Circular Coded Routing Algorithm .....	22
2.4 Fault-Tolerant Routing Algorithm under Single Link/Node Failure.....	23
2.4.1 Fault Model.....	23
2.4.2 Fault-Tolerant Routing Algorithm under Single Link/Node Failure.....	24
CHAPTER 3 PRDT ROUTER DESIGN .....	28
3.1 Data Units .....	28
3.2 PRDT Router Design .....	30
3.3 Building Blocks of Router .....	32
3.3.1 Input Channel Module .....	33
3.3.1.1 Input Flow Controller (IFC).....	34
3.3.1.2 FIFO .....	34
3.3.1.3 Input Controller (IC).....	34
3.3.2 Output Channel Module.....	36

3.3.2.1 Output Flow Control (OFC): .....	36
3.3.2.2 Output Controller (OC):.....	37
3.3.3 Crossbar Switch .....	37
3.3.4 Parameterized Round Robin Arbiter based Scheduler (PRRAS) .....	41
3.4 Design of PRRAS .....	41
3.4.1 Decoder .....	43
3.4.2 PRRA .....	44
3.4.2.1 <i>i</i> -node .....	45
3.4.2.2 <i>r</i> -node .....	46
3.4.2.3 <i>l</i> -node .....	46
3.4.2.3 PRRA Implementation.....	47
CHAPTER 4 OPTIMIZATION AND SIMULATION RESULTS.....	48
4.1 Results of components of the router .....	48
4.1.1 Input Channel Module .....	48
4.1.2 Output Channel Module.....	51
4.1.3 Scheduler.....	52
4.2 Results of Complete Router .....	54
4.3 Report on PRDT Network .....	56
4.4 Timing Analysis of the 4x4 PRDT network: .....	58
CHAPTER 5 STUDY OF CROSSBAR SWITCH DESIGN .....	60
5.1 Overview of Two Types of Crossbar Switch Design .....	60
5.2 Designs in Verilog Code .....	61
5.3 Design using Layout Editor .....	62
CHAPTER 6 CONCLUSION AND FUTURE WORK .....	69
6.1 Conclusion .....	69
6.2 Future Work .....	70
REFERENCES .....	71
APPENDIX.....	79
VITA .....	85



## LIST OF FIGURES

Figure 1.1:	A mesh-based NoC architecture.....	2
Figure 1.2:	Regular network topology.....	4
Figure 1.3:	Irregular topologies. ....	5
Figure 2.1:	PRDT structures ((a) 8x8, (b) 4x4) .....	16
Figure 2.2:	Directions for dimensions. ....	18
Figure 2.3:	Shift code representation .....	20
Figure 2.4:	Single link failure on rank- $r$ torus of the RDT(2, 2, 1)/ $\alpha$ structure .....	24
Figure 2.5:	Single node failure on rank- $r$ torus of the RDT(2, 2, 1)/ $\alpha$ structure .....	24
Figure 3.1:	Packet format.....	28
Figure 3.2:	Flit format.....	29
Figure 3.3:	Header flit format. ....	29
Figure 3.4:	Communication ports of the router. ....	30
Figure 3.5:	Function flow diagram of router. ....	32
Figure 3.6:	Block diagram of PRDT router. ....	33
Figure 3.7:	Input controller.....	33
Figure 3.8:	Directions of the output channels.....	35
Figure 3.9:	Output channel module. ....	36
Figure 3.10:	Crossbar structure.....	39
Figure 3.11:	Mux-based crossbar.....	40
Figure 3.12:	Block diagram of PRRAS. ....	42
Figure 3.13:	Complete Design of PRRAS. ....	43
Figure 3.14:	Structure of PRRA.....	45
Figure 3.15:	Structure of $l$ -node.....	47
Figure 4.1:	Area distribution of components in $ICM_L$ .....	50
Figure 4.2:	Area Distribution of All Components of the Router. ....	55
Figure 4.3:	Area and Timing Results of Two Routers (Normal and Optimized). ....	56
Figure 4.4:	Comparison of Power Results of Two Routers (Normal and Optimized). ....	56
Figure 4.5:	Calculation of $t_h$ w.r.t (0,0) as source. ....	59
Figure 5.1:	2-Input multiplexer based on transmission gates. ....	63
Figure 5.2:	Crossing switch designed with transmission gate. ....	63
Figure 5.3:	4x4 Crossbar switch using crossing switches. ....	64
Figure 5.4:	Area vs. Crossbar switch size.....	65
Figure 5.5:	Timing delay vs. Crossbar switch size.....	66
Figure 5.6:	Power consumption vs. Crossbar switch size.....	66
Figure 5.7:	Transistor count vs. Crossbar switch size. ....	67

## LIST OF TABLES

Table 2.1:	Comparison of four types of interconnection networks. ....	17
Table 4.1:	Results of area, timing, and power consumption of two types of IC <sub>L</sub> s.....	49
Table 4.2:	Results of area, timing, and power consumption of FIFO and IFC.....	49
Table 4.3:	Results of area, timing, and power consumption of ICMs with IC <sub>L</sub> .....	50
Table 4.4:	Results of area, timing, and power consumption of ICMs with IC <sub>o</sub> .....	51
Table 4.5:	Results of area, timing, and power consumption of OCM.....	51
Table 4.6:	Area results (in number sq microns) of two PRRA designs.....	52
Table 4.7:	Timing results( <i>ns</i> ) of two PRRA designs. ....	53
Table 4.8:	Results of area, timing, and power consumption of PRRAS with 9 inputs..	54
Table 4.9:	Results of area, timing, and power consumption of two routers .....	55
Table 4.10:	Results of 4x4 PRDT Network Using Routers with CCVR Algorithm. ....	57
Table 4.11:	Results of 4x4 PRDT Network Using Routers with VR Algorithm.....	58
Table 5.1:	Results of MUX-based 9x9 crossbar switch design. ....	62
Table 5.2:	Comparison of MUX-based and crosspoint-based crossbar designs.....	62
Table 5.3:	Report on crossbar (using layout).....	65

## ACKNOWLEDGEMENTS

I would like to thank Dr. Mei Yang, my mentor, for giving me the opportunity to have this research experience and providing assistance and support to present this work.

I would also like to thank the professors in my committee: Dr. Venkatesan Muthukumar for setting up and helping me in SYNOPSIS tools and Verilog coding, Dr. Yingtao Jiang for valuable comments and suggestions on VLSI physical design and Dr. Yoohwan Kim for agreeing to be on the thesis committee. I would also want to thank my family and friends for their support and encouragement during my Masters.

## CHAPTER 1

### INTRODUCTION

As predicted by the International Technology Roadmap for Semiconductors (ITRS) [32], for the next 5 to 10 years, System-on-Chips (SoCs), using 32 nm transistors operating below one volt, will grow to multi-billion transistors running at a frequency of 10GHz or higher. One of the major challenges in designing such highly integrated SoCs will be to find an effective way to integrate pre-designed Intellectual Property (IP) cores for power and performance concerns [2]. As the device feature size is continuously shrinking and the bandwidth requirements are increasing, traditional bus-based SoC architecture [42] have been found creating a performance bottleneck. Networks-on-Chip (NoC) communication architectures have emerged as a promising alternative to overcome those limitations of bus-based communication infrastructure by employing a packet-based micro-network for inter-IP communication.

As the interface of an IP to the on-chip interconnection network, the router design has an important impact to the cost and performance a NoC design. This thesis is focused on the design and implementation of a parameterized NoC router. This chapter introduces the background of this work and gives the outline of the thesis. First, an overview of NoC architectures is given and challenges in NoC designs are addressed. Next, the review of existing router design is provided. At the end of this chapter, we discuss the motivation for this study followed by an outline of the thesis.

## 1.1 Overview of NoC Architecture

In general, a packet-based NoC consists of routers, the network interface between the routers and the processing units, and the interconnection network [15]. Figure 1.1 shows a regular 4x4 NoC architecture with a mesh-based interconnection network. Each processing unit can be a general-purpose processor, a DSP, an embedded memory etc. Each processing unit is attached to a router which connects it to its neighboring processing units. Later in the text, we use *node* to refer a processing unit and its associated router.

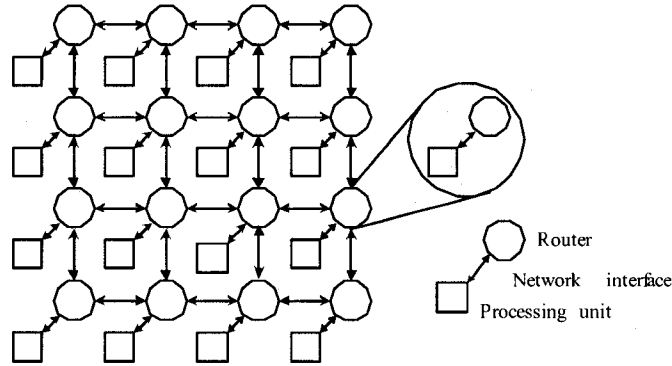


Figure 1.1: A mesh-based NoC architecture.

The design of a NoC system must address the following challenges.

- *Scalability*: In a NoC system, the interconnection network plays an important role in providing scalability to accommodate larger number of transistors and alleviate design productivity gap [24] [19]. On-chip networks will likely use networks with lower dimensionality to keep wire lengths short [30].
- *Energy-efficiency*: When designing a NoC system, power needs to be treated as a major design constraint as significant amount of power is consumed by its

interconnection network [73]. The power consumption of the interconnection network largely depends on the energy consumed by the routers and the energy consumed on the interconnection links, which is related to the interconnection architecture itself, the routing and switching schemes employed, and the implementation techniques [24].

- *Reconfigurability*: Reconfigurable architecture emerges as one of the most important architectural paradigm for satisfying the simultaneous requirements for application performance and flexibility [33] [73]. Reconfigurable architecture is particularly desirable for real-time applications due to the high-performance they can offer, the cost saved, improved time-to-market, and improved flexibility and upgradability [38].

The design of NoCs involves trades-off between several important choices, such as topology selection [47] [1] [29] [35] [48] , communication protocol selection [20] [26], and application mapping to processing units [34] [50]. A formal categorization of the NoC design issues is given in [53]. In the following, an overview of the topologies and communication protocols used in NoCs is provided as they are directly related to the router design.

#### 1.1.1 Network Topology

Most NoCs adopt regular forms of network topologies that can be laid out on a chip surface (a 2-dimensional plane), for example,  $k$ -ary 2-cube, commonly known as grid-type topology. The  $k$ -ary  $n$ -cube topology, where  $k$  is the degree of each dimension and  $n$  is the number of dimensions, was first described in [10] for multicomputer networks. The popular  $k$ -ary 2-cube type NoC topologies are the mesh which uses bidirectional links and torus which uses unidirectional links. To reduce the routing delay on long wires, fold

torus is proposed in [11]. The  $k$ -ary tree and the  $k$ -ary  $n$ -dimensional fat tree are two alternate regular forms of networks explored for NoC . Figure 1.2 shows examples of regular forms of topology. The network area and power consumption scales predictably for increasing size of regular forms of topology. Generally, mesh topology makes better use of links (utilization) [45], while tree-based topologies are useful for exploiting locality of traffic. In [37] , Octagon NoC is another example for novel regular NoC topology.

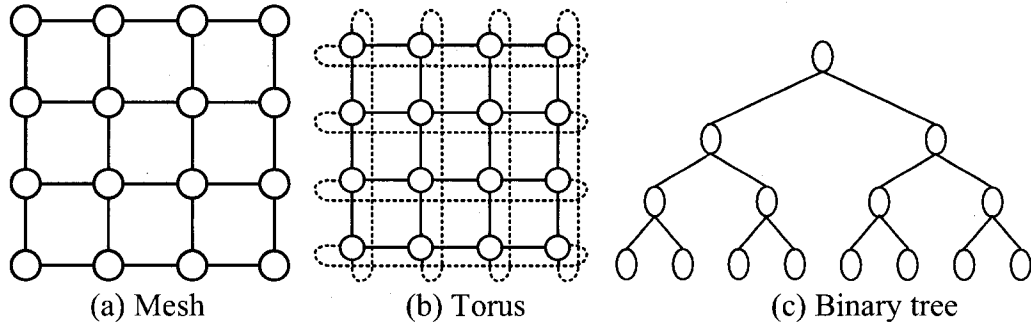


Figure 1.2: Regular network topology.

Irregular forms of topologies are derived by mixing different forms in a hierarchical, hybrid, or asymmetric fashion [5], which are usually based on the concept of clustering. In [55], the impact of clustering on five NoC topologies is studied. Irregular forms of topologies scale nonlinearly in regard to area and power consumption [5]. The examples of indirect tree-based networks are fat-tree in SPIN [21] and butterfly in [55]. The fat-tree (figure 1.3) used in SPIN is proven in to be the most hardware efficient compared to other networks [21]. However, the size of this network grows in  $(n \log n)/8$ , where  $n$  is the number of terminals. In [22], a honeycomb structure is proposed. As shown in Figure 1.3 (a), in the honeycomb NoC, the resources including computational, storage and I/O are

organized as nodes of the hexagon with a local switch at the center that interconnects these resources.

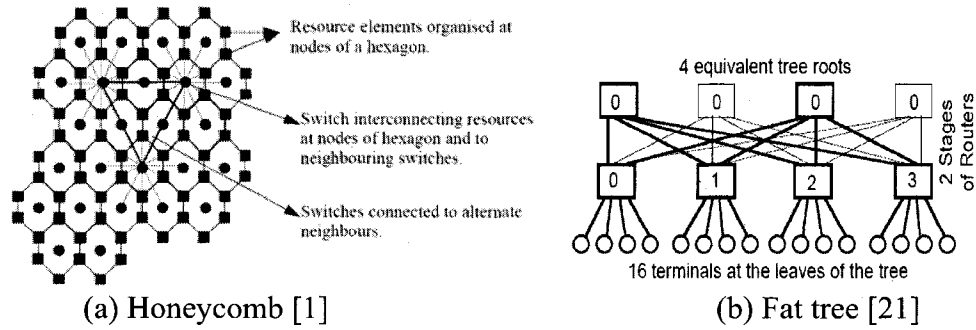


Figure 1.3: Irregular topologies.

The major problems with some of the aforementioned topologies are: they are either not scalable (e.g., mesh and torus), or not reconfigurable (most irregular topologies). To address these two problems, a novel class of topologies named Recursive Diagonal Torus (RDT) is proposed for NoC [80]. The RDT structure is constructed by recursively overlaying 2-D diagonal torus, and it was originally designed as the interconnection network of a massively parallel processor [77][78][69]. In [80], it shows that the RDT structure has the following features: recursive structure, smaller diameter and average distance, embedded mesh/torus topology, a constant node degree of 8, and robust routing schemes. Hence, it has good scalability and is reconfigurable to simpler structures (such as mesh or torus). A special type of RDT, called Perfect RDT (PRDT) [75], is considered to be a promising on-chip interconnection network topology due to its symmetric structure and simpler link connections than other RDT structures.



### 1.1.2 Routing Algorithm

The selection of routing scheme greatly affects the network performance [66] and power consumption [28][54]. Given a NoC architecture and the source and destination nodes, the routing algorithm running in each router decides the output port to route the packet. Implementation complexity and performance requirements are two major concerns in selecting the routing algorithm [53]. In general, the routing algorithms for NoC can be classified into two categories – deterministic routing and adaptive routing [1].

In deterministic routing, the routing algorithm is independent of the network conditions. Hence, it requires fewer resources and guarantees an orderly packet arrival.

XY (or YX) routing is deterministic routing algorithm wherein a packet is first forwarded in the X dimension and then along the Y dimension, restricting the maximum number of allowed turns to one [10]. An extension to this algorithm has been proposed in [20][71], which imposes certain turn rules on the XY routing algorithm. Deflection routing is another deterministic routing algorithm that forwards the packet towards the path with the lowest delay [6]. The odd-even turn model [7] is designed for partially adaptive wormhole routing algorithms without adding virtual channels. In comparison with the well-known turn model, this scheme provides more even routing adaptiveness. The model restricts the locations where some turns can be taken so that deadlock is avoided. The degree of routing adaptiveness provided by the model is more even for different source-destination pairs. The mesh network may benefit from this feature in terms of communication efficiency. In addition, this property results in a smaller fluctuation of the network performance with respect to different traffic patterns.

In adaptive routing, the path that a packet chooses depends on the source and destination address as well on the dynamic traffic conditions. Hence, it may provide better throughput and lower latency by allowing alternate paths based on the network congestion. A contention aware hot potato routing scheme is proposed in [52]. A variation to the model developed in [7], where an odd-even adaptive routing algorithm for meshes is proposed. Hu *et al.* [74], propose a routing scheme which switches between deterministic and adaptive according to network congestion situation. There has been an in-depth survey on some efficient routing algorithms in [49]. Comparison of various routing algorithms over different topologies is discussed in [13][23] [51].

Deterministic routing requires less resource and guarantees 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. Out-of-order message arrival is an important problem associated with adaptive algorithms. Deadlock is an important issue in NOC's, since deadlock (livelock) detection and recovery mechanisms are expensive and they may lead to unpredictable delays. Virtual channels may be used to avoid deadlocks as well as to utilize the channel bandwidth better. Deterministic and partially adaptive algorithms based on the turn model [20]; guarantee free deadlock and livelock operation, while fully adaptive strategies require extra precaution and virtual channels. Deterministic routing is appropriate if the traffic generated by the application under consideration is predictable.

Being application-dependent, the routing algorithm for NoCs can indeed be customized to match the application traffic pattern [53]. Stochastic routing for fault-tolerant in NoCs has been discussed in [16]. In [76], the fault-tolerant routing scheme for

RDT-based NoCs is discussed. A power-aware, adaptive routing strategy that regulates the routing decisions to satisfy peak power constraints is proposed in [63]. However, this approach does not address timing constraints, which are likely to coexist with power constraints. Hence, a power- and performance aware- technique [28][29] is needed . Moreover, more complicated routing strategies result in larger design.

### 1.1.3 Switching Technique

A problem related to routing is the switching technique used in the network, which determines when the routing decisions are made, how the switches inside the routers are set/reset, and how the packets are transferred along the switches [53].

Switching techniques have been a well researched area in traditional data networks for a long time. There are four switching techniques which are considered promising for NoCs.

- Store-and-forward: Commonly known as packet switching, the entire packet is stored in the buffer at an intermediate node before it is forwarded to a selected neighboring node based on the destination node address stored in the packet header. In packet switching, the bandwidth is utilized in a flexible way. As an example, the CLICHÉ NoC [41] employs store-and-forward switching.
- Circuit switching: Circuit switching involves the establishment of a physical circuit between the source and destination nodes and reservation of the circuit until the transport of data is complete. As such, circuit switching can provide guaranteed service as required by some applications.
- Wormhole: This technique combines the advantages of packet switching and circuit switching and achieves low data latency. In wormhole switching [9][49],a

packet is divided into fixed size flits (flow control unit, which is typically set as one or multiple wire bit width) and then these flits are routed through the network one after another, in a pipelined fashion. The header flit of a packet contains the destination address and other control information. At each intermediate node, once receiving a header flit, it makes the routing decision and sets up the connection from the incoming input port to the destined output port in the switch. This connection will be valid until the last flit of the packet is transmitted. Due to the small flit size, wormhole switching achieves low data latency with small buffer requirement.

- Virtual cut-through (VCT) [39]: In this switching technique, the forwarding router waits for a guarantee from the next node in the path that it will accept the entire packet. This handshaking allows the forwarding router to transmit the intermediate flits as it receives them, thus reducing the data latency.

Among the commonly used switching techniques, wormhole switching seems to be the most promising one for typical NoC applications due to its advantage of low data latency with small buffer requirement. A wormhole NoC router [49] is used in MANGO [3]. In [81], a soft core router, RASoC, is developed which uses wormhole switching.

In data networks, wormhole switching is preferred than circuit switching due to the poor performance of the latter under dynamic traffic. However, for application-specific NoCs, circuit switching is preferred. Moreover, guaranteed service operation, as required by some applications, is relatively easier to be satisfied by using circuit switching. For example, the NOSTROM NoC [45] adopts circuit switching and implements a service of guaranteed bandwidth (GB) using virtual circuits [4].

Therefore, circuit switching is a promising alternative, despite its implementation complexity and static nature. It remains to be seen whether or not a particular switching technique, or a hybrid combination, is more advantageous. Some of the hybrid switching techniques are given in [25][65] [14].

The switch used in the router is the device that sends flits from input ports to output ports. Its size is determined by the number of input/output ports. In [70], the estimation for power consumption for various switch designs is discussed. Bit energy consumed by the switching fabric, internal buffers, and interconnect wires is calculated and analyzed for several switching fabrics, including crossbar, fully connected, banyan, and batcher-banyan networks.

#### 1.1.4 Buffering Technique

The next important aspect to be discussed is the buffer allocation. The input channel buffers at each router in the NoC have a serious impact on the overall area. For instance, by increasing the buffer size at each input channel from 2 to 3 words, the router area of a 4x4 NoC increases by 30% or more [61]. Thus, the overall use of buffering resources has to be minimized to reduce the implementation overhead in NoCs. On the other hand, depending on the network load, increasing the buffer size can reduce the data latency by orders of magnitude. The properties of on-chip buffers are studied in [61]. The authors report gate-level area estimates and analyze the performance of the network and buffers utilization across the network. An efficient algorithm for the buffer size allocation problem is proposed in [27]. Although queuing theory can help achieving significant performance improvements through smart buffer allocation, many problems remain to be solved. Some critical issues are discussed in [27].

## 1.2 Existing NoC Router Design

In the literature, a number of NoC router designs have been proposed. In the following, a review of these designs is given.

A router based on adaptive routing is proposed [40] with minimum message latency. It is a two-stage pipelined architecture; using look-ahead routing, speculative allocation and optimal output path selection, in case of concurrency. One more novelty of this router is the use of decomposed cross-bar switches, which reduces the contentions.

In [43], a router based on the new crossbar scheduling algorithm called TREE is proposed. The Tree algorithm has many advantages, such as the arbitration is computed concurrently with packet propagation and thereby latency is reduced, and the overall area of the scheduler is reduced compared with the round-robin scheduling algorithm. The complexity of the TREE algorithm is  $O(\log n)$  whereas that of round-robin is  $O(n)$  for a  $n$ -input/output crossbar switch.

In [59], the guaranteed throughput (GT) and best effort (BE) router architectures are combined in an efficient implementation by sharing resources. Based on circuit switching, the GT router uses slot table to avoid contentions on a link and for dividing the bandwidth per link. Based on packet switching, the BE router uses matrix scheduling. In this design, the GT and BE routers are combined together and the traffic is controlled by an arbitration unit and the two routers share the links and switch in common.

In [62], a prototype design of a 5-input/output scalable switching node is presented. The packet connected circuit (PCC) technique, a combination of circuit switching and packet switching is used in this design. The switching node basically consists of input

and output finite state machine (FSM), priority encoder, address decoder module, and an arbiter.

In [25], a switch is designed without using memory buffer. Here the PCC technique is also used but without any arbiter. This switch is implemented with lots of design simplifications in AMS 0.18 technology.

In [81][83], a soft router core, router architecture for SoC (RASoC), is developed using parameterized VHDL model. The RASoC features a distributed router architecture based on wormhole switching approach. It uses XY routing algorithm (deterministic), round-robin arbitration, and input buffering. The advantage of this router design is that, it is been implemented in parameterized VHDL so that it can be reused for different sizes in order to meet the requirements of the target applications by just changing the parameters. RASoC is implemented on SoCIN (System-on-Chip Interconnection network).

In [82], PARIS (Parameterizable Interconnection Switch), advancement to RASoC is proposed. PARIS has all the benefits of RASoC and extends the parameterization to the techniques used for packet forwarding (eg. Routing, arbitration and flow control). PARIS is implemented on SoCINfp (System-on-Chip Interconnection network fully parametrized), an advancement of SoCIN.

In [60], asynchronous multi-service level QNoC router is proposed. Wormhole routing and a simplified version of source specified routing are used in this design. A MUTEX-NET arbiter is adopted and its fairness is discussed in the paper.

In [3], the MANGO router is proposed. This is a clockless router which is derived from the globally asynchronous and locally synchronous (GALS) principle. Some advantages using the clockless NoC are; they operate in maximum speed and they have

zero dynamic power consumption. The MANGO router exploits virtual channels (VCs) to provide connection-oriented guaranteed services (GS) and connectionless best effort (BE) routing. The BE router uses source routing and the GS router is setup by programming through the BE router. The GS router uses a VC control module which employs share-based VC control, which is nothing but a non-blocking circuit switching. In [57][66] some techniques for synthesis of custom NoC architectures are explained.

### 1.3 Contribution and Overview of the Thesis

As discussed in previous section, the Recursive Diagonal Torus (RDT) is a class of topologies suitable for NoCs. Particularly, the PRDT structure has reduced complexity but keeps the distinct architectural features of RDT. The study in [75] shows that the PRDT-based is promising and it is feasible to be implemented with current VLSI technologies.

Discussed in Section 1.1., the cost and performance of a NoC largely depends on the router architecture. Hence, it is necessary to build an efficient router for PRDT-based NoC. This thesis is focused on developing an efficient parameterized router for PRDT-based NoC. Due to the architecture feature of PRDT, the same router can also be applied for mesh/torus-based NoCs. Two routing algorithms and wormhole switching technique are implemented in this router.

The rest of the thesis is organized as follows. In Chapter 2, the PRDT structure and the vector routing algorithm will be introduced in details. In Chapter 3, the design of router will be described. In Chapter 4, the experiment results will be provided and



discussed. In Chapter 5, a comparative study on Crossbar is presented. Chapter 6 concludes the thesis.

## CHAPTER 2

### PRDT AND ITS ROUTING ALGORITHMS

#### 2.1 Introduction

Among the various interconnection networks proposed for NoCs (refer to Chapter 1), the RDT structure [77] has the following advantages: 1) high scalability with its recursive structure, 2) small diameter and average distance, 3) architectural reconfigurability with its embedded mesh/torus topology, and 4) fault-tolerance capability with a constant node degree and robust routing schemes. In [80], it is shown that the RDT structure is feasible to be implemented with current VLSI technologies. A special type of RDT, PRDT is studied in [75]. PRDT has a simpler structure but keeps the architectural features of RDT. Hence, it is suitable to build on-chip interconnection network for NoCs with several to hundreds of processing units (interchangeably with nodes).

In this chapter, the structures of RDT and PRDT will be described in details followed by the description of the routing algorithms designed for PRDT.

#### 2.2 Structure of RDT and PRDT

The RDT structure is constructed by recursively overlaying 2-D diagonal meshes (tori) [77][78]. The *base torus* is a two-dimensional square array of nodes, each of which is numbered with a two-dimensional number  $(i, j)$ ,  $0 \leq i \leq N-1$ ,  $0 \leq j \leq N-1$ , where  $N = nk$  and both  $n$  and  $k$  are natural numbers. The torus network is formed with four links between

node  $(x, y)$  and its neighboring four nodes:  $(\text{mod}(x \pm 1, N), y)$  and  $(x, \text{mod}(y \pm 1, N))$ . The base torus is also called *rank-0 torus*. On top of rank-0 torus, a new torus-like network (*rank-1 torus*) is formed by adding four links between node  $(x, y)$  and nodes  $(x \pm n, y \pm n)$ . The direction of the new torus like network is at an angle of 45 degrees to the original torus. On rank-1 torus, another torus-like network (*rank-2 torus*) can be formed by adding four links in the same manner. In a more general sense, a *rank-(r+1) torus* can be formed upon *rank-r torus*. In [80], one type of RDT structure,  $\text{RDT}(2, 2, 1)/\alpha$ , is studied. It is demonstrated that the least number of layers needed for laying out  $\text{RDT}(2, 2, 1)/\alpha$  is 6, which is feasible for implementation with current VLSI technologies. Hence,  $\text{RDT}(2, 2, 1)/\alpha$  offers a practical solution for on-chip interconnection network, especially for large-scale NoC systems.

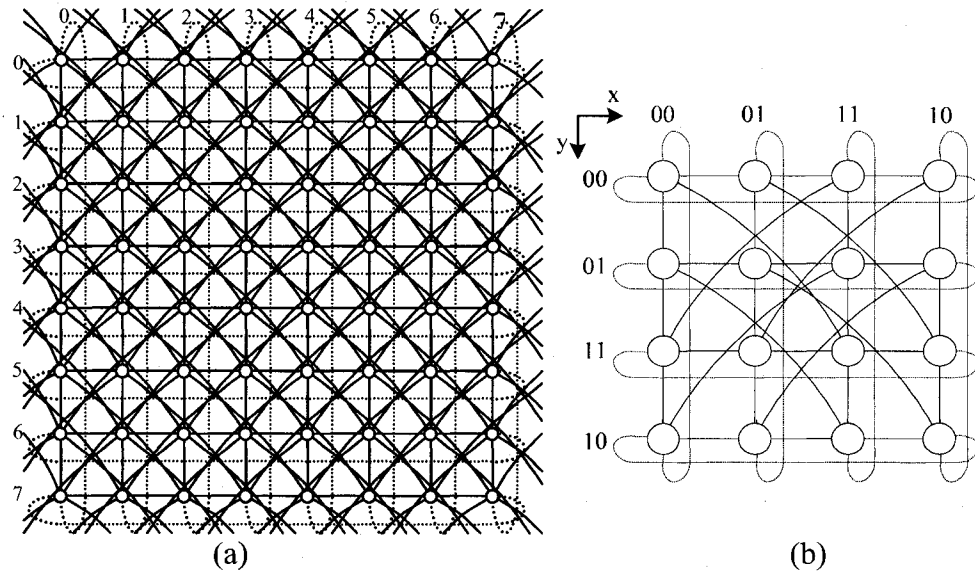


Figure 2.1: PRDT structures ((a) 8x8, (b) 4x4) [75].

A perfect RDT [75] is an RDT in which every node has links to form all possible upper rank tori (i.e.  $\text{RDT}(n, R, R)$ ), denoted as  $\text{PRDT}(n, R)$ , where  $n$  is the cardinal

number, and  $R$  is the maximum rank. Particularly, we consider PRDT(2, 1), in which each node has a constant degree of 8 except for PRDT(2, 1) with 4x4 nodes. Figure 2.1(a) shows the structure of 8x8 PRDT (2, 1). Figure 2.1(b) shows the structure of 4x4 PRDT (2,1), where each node has 5 links, one on the rank-1 torus and the other four on the rank-0 torus.

Table 2.1 [75] shows the comparison of the diameter and average distance of PRDT (2, 1) , mesh, torus, and hypercube with different network sizes. One can see that PRDT (2, 1) has the smallest diameter and average distance among the four different structures for most network sizes. With only rank-0 and rank-1 links, the wiring cost of PRDT (2, 1) is dramatically reduced compared with RDT (2, 2, 1)/ $\alpha$ . Hence, PRDT (2, 1) is very promising for interconnecting NoC systems with tens to hundreds of nodes.

Table 2.1: Comparison of four types of interconnection networks. (R stands for diameter, AD stands for average distance.) [75]

	PRDT(2,1)		Mesh		Torus		Hypercube	
Size	R	AD	R	AD	R	AD	R	AD
4x4	2	1.56	6	2.67	4	2	4	2
8x8	3	2.28	14	5.33	8	4	6	3
16x16	5	3.64	30	10.67	16	8	8	4
32x32	9	6.31	62	21.33	32	16	10	5

### 2.3 Routing Algorithm for PRDT

XY routing is implemented in PRDT. Two variations of vector XY routing algorithm are used. The first one is normal vector routing [67] and the second is circular coded vector routing, which is derived from the binary routing algorithm [12] for PRDT.

### 2.3.1 Vector Routing Algorithm (VR)

The basic routing algorithm for PRDT (2, 1) is the vector routing algorithm [67], in which a route from a source node to a destination node is represented with a vector. The goal of the vector routing algorithm is to represent the vector with an expression that combines all unit vectors in the rank-0 and rank-1 torus. Figure 2.2 shows the directions of the unit vector for each rank torus, which rotate in clockwise direction at an angle of 45 degrees as the rank increases.

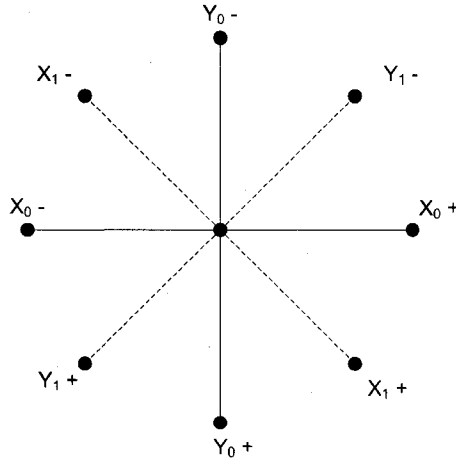


Figure 2.2: Directions for dimensions.

The vector from a source node to a destination node is denoted as  $\vec{A}$ ,  $\vec{A} = a\vec{X}_0 + b\vec{Y}_0$ , where  $\vec{X}_0$  and  $\vec{Y}_0$  are the unit vectors of the rank-0 torus.  $\vec{A}$  can be derived as  $\vec{A} = a_1\vec{X}_1 + b_1\vec{Y}_1 + a_0\vec{X}_0 + b_0\vec{Y}_0$ , where  $\vec{X}_1$  and  $\vec{Y}_1$  are the unit vectors in the rank-1 torus, and  $a_1$ ,  $b_1$ ,  $a_0$  and  $b_0$  are chosen such that the hop number on the route (which is determined by  $a_1 + b_1 + a_0 + b_0$ ) is minimized.  $\vec{A}$  can be represented with a combination of the unit vectors  $\vec{X}_R$ ,  $\vec{Y}_R$ , ...,  $\vec{X}_0$ ,  $\vec{Y}_0$  on rank- $R$  to rank-0 tori as

$$\vec{A} = a\vec{X}_0 + b\vec{Y}_0 = v_{Rh}\vec{X}_R + v_{Rv}\vec{Y}_R + \dots + v_{rh}\vec{X}_r + v_{rv}\vec{Y}_r + \dots + v_{0h}\vec{X}_0 + v_{0v}\vec{Y}_0, \quad \text{where } (v_{rh}, v_{rv})$$

represents the vector on rank- $r$  torus, where  $R \geq r \geq 0$ . And  $v_{rh}$  and  $v_{rv}$  are maximized in order to use the upper rank torus as much as possible. Given the vector  $\vec{A} = a\vec{X}_0 + b\vec{Y}_0$  corresponding to the destination address  $(a, b)$ ; the general vector routing algorithm is given as follows, where the array *vector* is used to store the routing vectors for each rank.

Algorithm Vector Routing for PRDT( $n, R$ ):

```

begin
  loop  $r = R$  downto 0
     $g = (a+b)/2n$ 
     $f = (b-a)/2n$ 
     $v_{rh} = a - (n*g - n*f)$ 
     $v_{rv} = b - (n*g + n*f)$ 
     $a = g$ 
     $b = f$ 
  endloop
end

```

The vectors are computed at the source node and encapsulated into the packet. Then the packet is routed following the order of rank- $R$ , rank- $(R-1)$ , ... rank-0 vectors. On the same rank torus, the routing is performed according to a predetermined order, for example, XY routing [49].

For PRDT (2, 1), only 2 ranks are involved, so the algorithm can be simplified. As shown below, the routing vectors are directly stored in 4 variables ( $vecX_0$ ,  $vecY_0$ ,  $vecX_1$ ,  $vecY_1$ ) and no iteration is required.

Vector Routing Algorithm for PRDT(2, 1):

```

begin
   $vecX1 = (a + b) / 2n$ 
   $vecY1 = (b - a) / 2n$ 
   $vecX0 = a - (n * vecX1 - n * vecY1)$ 
   $vecY0 = b - (n * vecX1 + n * vecY1)$ 
end

```

At each intermediate node, the routing vectors will be checked in the order of  $vecX_1$ ,  $vecY_1$ ,  $vecX_0$ ,  $vecY_0$ . The packet will be routed to the direction decided by the first variable with non-zero and the variable's value will be updated accordingly.

### 2.3.2 Circular Coded Vector Routing (CCVR)

CCVR (unpublished report) is an extension of the binary routing algorithm [12]. The X coordinate and the Y coordinate of the nodes are represented with a 2-bit binary shift code, as shown below. The binary code for each node is a 4-bit binary number combining its X coordinate and Y coordinate. The codes for all the 16 nodes of 4x4 network are listed below.

0000	0100	1100	1000
0001	0101	1101	1001
0011	0111	1111	1011
0010	0110	1110	1010

Figure 2.3: Shift code representation

#### 2.3.2.1 Basic functions of shift-code

At the source node, the  $f_r$  function is used to determine the direction ( $c$ ) of routing. Given two  $m$  bits shift-code:  $A$  and  $B$ , where  $A = a_1a_2a_3\dots a_m$  and  $B = b_1b_2b_3\dots b_m$ , the

function  $f_r$  is defined as  $c = f_r(A, B)$  as below, where  $p = \sum_{i=1}^m (a_i \oplus b_i)$  gives the distance

between  $A$  and  $B$ :

If  $p = 0$ , then  $c = 0$

if  $a_i = b_i = 0$ , then

if  $a_i = b_i = 1$  then

$$c = \begin{cases} 1 & \text{if } \sum_{i=1}^m b_i > \sum_{i=1}^m a_i \\ -1 & \text{if } \sum_{i=1}^m b_i < \sum_{i=1}^m a_i \end{cases} \quad c = \begin{cases} 1 & \text{if } \sum_{i=1}^m b_i < \sum_{i=1}^m a_i \\ -1 & \text{if } \sum_{i=1}^m b_i > \sum_{i=1}^m a_i \end{cases}$$

if  $a_i = 0, b_i = 1$ , then

if  $a_i = 1, b_i = 0$ , then

$$c = \begin{cases} 1 & \text{if } \sum_{i=1}^m \bar{b}_i < \sum_{i=1}^m a_i \\ -1 & \text{if } \sum_{i=1}^m \bar{b}_i > \sum_{i=1}^m a_i \end{cases} \quad c = \begin{cases} 1 & \text{if } \sum_{i=1}^m \bar{b}_i > \sum_{i=1}^m a_i \\ -1 & \text{if } \sum_{i=1}^m \bar{b}_i < \sum_{i=1}^m a_i \end{cases}$$

Given the source and destination addresses  $S=S_xS_y$ ,  $D=D_xD_y$ , where  $S_x/D_x$  and  $S_y/D_y$  represent the  $X$  coordinate and  $Y$  coordinate of the source/destination node in  $m$ -bit shift code, respectively, the direction and distance values on both coordinates,  $(c_x, p_x)$  and  $(c_y, p_y)$ , can be calculated according to the above formula.

### 2.3.2.2 Intermediate node function $f_s$

Once the direction values and distance values are calculated, the intermediate node's address  $M_xM_y$  for  $N \times N$  PRDT(2, 1), which is used in Circular Coded Vector routing algorithm, is calculated as below,

$$M_x = f_s(S_x, c_x, N) = \begin{cases} \text{mod}(S_x + n, N) & \text{when } c_x \geq 0 \\ \text{mod}(S_x - n, N) & \text{when } c_x < 0 \end{cases}$$

$$M_y = f_s(S_y, c_y, N) = \begin{cases} \text{mod}(S_y + n, N) & \text{when } c_y \geq 0 \\ \text{mod}(S_y - n, N) & \text{when } c_y < 0 \end{cases}$$

Where  $c_x$  and  $c_y$  represents direction along horizontal(X) and vertical(Y) co-ordinates.



### 2.3.2.3 Circular Coded Routing Algorithm

In Circular Coded Vector routing, all the routing steps are calculated and stored in a routing vector ( $vecX_0, vecY_0, vecX_1, vecY_1$ ) based on the destination node address  $D$  and the source node address  $S$ . The routing vector is then embedded in the data packet. In the transmission of the data packet, the routing direction in each step will be decided according to the value in the routing vector. Below lists the pseudo code for the circular coded routing algorithm for PRDT(2, 1).

Circular Coded Routing Algorithm for PRDT(2, 1):

```
begin
  initialize  $vecX_0, vecY_0, vecX_1, vecY_1$  to zero
  let  $M_x=S_x, M_y=S_y$ 
  while ( $D_x \neq M_x$  and  $D_y \neq M_y$ ) do
    calculate  $(c_x, p_x)$  and  $(c_y, p_y)$  using  $f_r$  function
    if ( $c_x = 0$  and  $c_y = 0$ ) then
      assign all vectors as zero
    elseif ( $p_x + p_y \leq 2$ ) then //do rank 0 routing
      if ( $c_x \geq 0$ ) then  $vecX_0 = vecX_0 + 1$ 
      if ( $c_x < 0$ ) then  $vecX_0 = vecX_0 - 1$ 
      if ( $c_y \geq 0$ ) then  $vecY_0 = vecY_0 + 1$ 
      if ( $c_y < 0$ ) then  $vecY_0 = vecY_0 - 1$ 
      find  $M_x$  and  $M_y$  using  $f_s$  function
    elseif ( $p_x + p_y > 2$ ) then //do rank 1 routing
      if ( $c_x \geq 0$  and  $c_y \geq 0$ ) then  $vecX_1 = vecX_1 + 1$ 
      if ( $c_x \geq 0$  and  $c_y < 0$ ) then  $vecY_1 = vecY_1 - 1$ 
      if ( $c_x < 0$  and  $c_y \geq 0$ ) then  $vecY_1 = vecY_1 + 1$ 
      if ( $c_x < 0$  and  $c_y < 0$ ) then  $vecX_1 = vecX_1 - 1$ 
      find  $M_x$  and  $M_y$  using  $f_s$  function
  endif
```

**endwhile**  
**end**

For PRDT(2, 1), the while loop at most executes twice to get the routing vector calculated.

## 2.4 Fault-Tolerant Routing Algorithm under Single Link/Node Failure

The major cause affecting the reliability of the VLSI global interconnects is the shrinking of the feature size [32], which exposes them to different faults of permanent, transient or intermittent nature. This degrades NoC's QoS characteristics [18] or, eventually, led to failures of the whole NoC-based system. Traditionally, error detection and correction mechanisms are used to protect communication subsystems against the effects of transient malfunctions. Fault-tolerant design of Network-on-chip communication architectures requires the addressing of issues pertaining to different elements described at different levels of design abstraction – these may be specific to architecture, interconnection, communication and application issues. In [76] various fault-tolerant routing schemes applicable to the RDT(2,2,1)/ $\alpha$ -based interconnection network is proposed. Though the fault models and fault-tolerant algorithm are designed for RDT(2, 2, 1)/ $\alpha$  structure, they are applicable for PRDT(2, 1). In the following paragraphs fault model and fault tolerant routing algorithm [76] are discussed. Only the single fault cases are considered.

### 2.4.1 Fault Model

The following assumptions are used in [76][79]. 1) Any link or node in the network can fail, and the faulty components are unusable; that is, data will not be transmitted over

a faulty link or routed through a faulty node. 2) The fault model is static, that is, no new faults occur during a routing process. 3) Both source and destination nodes (on any rank torus) are fault-free. 4) The faults occur independently. 5) If a node fails, the four links associated with the node on rank- $r$  torus also fail. 6) Faulty link(s)/node(s) are known to all other nodes in the same rank. Two types of faults were discussed [79], link failure and node failure and are shown in the Figure 2.4 and 2.5, where faulty links are marked by X.

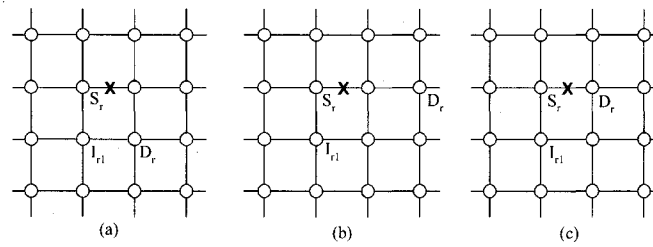


Figure 2.4: Single link failure on rank- $r$  torus of the RDT(2, 2, 1)/ $\alpha$  structure [76].

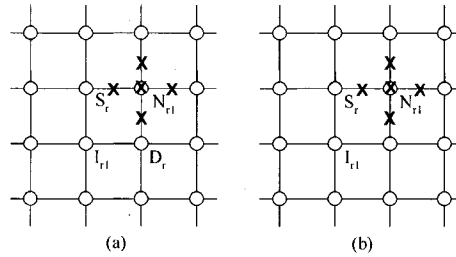


Figure 2.5: Single node failure on rank- $r$  torus of the RDT(2, 2, 1)/ $\alpha$  structure [76].

#### 2.4.2 Fault-Tolerant Routing Algorithm under Single Link/Node Failure

As one can see from Figures 2.4 and 2.5, if the failure is on the  $X$  (or  $Y$ ) direction, then the packet should be detoured by sending it through the  $Y$  (or  $X$ ) direction first. The vector needs to be changed to reflect the detour if necessary.

Let  $X_r+$  denote the  $X+$  direction on rank- $r$ ,  $X_r-$  denote the  $X-$  direction on rank- $r$ ,  $Y_r+$  denote the  $Y+$  direction on rank- $r$ , and  $Y_r-$  denote the  $Y-$  direction on rank- $r$ . A distributed fault-tolerant floating routing algorithm is listed as follows, where  $(i_x, i_y)$  represents the vector of a node  $i$  on rank-0 torus.

Vector Routing with Single Fault Tolerance Algorithm (FVRSF):

**begin**

// Step 1: The source node computes the vectors from rank R down to 0

// Other nodes decrement the vector value accordingly

**if**  $i = S$  &  $S \neq D$

    Call Vector Routing

$r = \max \{i \mid (vecX_i, vecY_i) \neq (0,0)\}$

**elseif** packet is received from  $X_r-$

$vecX_r = vecX_r - 1$

**elseif** packet is received from  $Y_r-$

$vecY_r = vecY_r - 1$

**elseif** packet is received from  $X_r+$

$vecX_r = vecX_r + 1$

**elseif** packet is received from  $Y_r+$

$vecY_r = vecY_r + 1$

**elseif** packet is received from  $X_0-$

$vecX_0 = vecX_0 - 1$

**elseif** packet is received from  $Y_0-$

$vecY_0 = vecY_0 - 1$

**elseif** packet is received from  $X_0+$

```

     $vecX_0 = vecX_0 + 1$ 

elseif packet is received from  $Y_0 +$ 
     $vecY_0 = vecY_0 + 1$ 

endif

// Step 2: Check the availability of the link and send the packets accordingly

if i is on rank-r
    if ( $vecX_r, vecY_r$ )  $\neq$  (0,0)
        if  $vecX_r \neq 0$  &  $vecX_r =$  not faulty
            send the packet to  $vecX_r$  direction
        elseif  $vecX_r = 0$  &  $vecY_r =$  not faulty
            send the packet to  $vecY_r$  direction
        elseif  $vecX_r \neq 0$  &  $vecX_r =$  faulty
            if  $vecY_r = 0$ 
                send the packet to  $Y_r +$ 
            else send the packet to  $vecY_r$  direction
            endif
        elseif  $vecY_r =$  faulty
            send the packet to  $X_r +$ 
        endif
    elseif find  $p = r - 1$  to 0 and ( $vecX_p, vecY_p$ )  $\neq$  (0,0) then
         $r = p$  , goto Step 2
    else quit
    endif
end

```

In FVRSF, the extra processing in case of failure is minimized. Another advantage of the FVRSF algorithm is that each router decides the best route based on the fault information of its four links on one rank torus. Thus no global fault information needs to

be maintained at each router. In this way, the overhead introduced is very low. In [76], an algorithm is proposed for multiple fault tolerance, but it is not implemented because of the hardware complexity involved.

## CHAPTER 3

### PRDT ROUTER DESIGN

In this chapter, the router designed for PRDT-based NoCs will be described in details. The data unit format will be introduced first before the router design is discussed.

#### 3.1 Data Units

In this router design, wormhole switching is implemented. It is assumed that the data messages to be sent will be first breaked into packets, which will be further decomposed into flits. As shown in Fig. 3.1, each packet is composed of header bits and payload/data bits. Header bits store the routing information, which includes the destination node address and routing vector (calculated by the routing algorithms). Each node in the network is identified by a pair of coordinates (xid, yid) in binary numbers, as shown in Figure 2.1b.

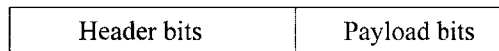


Figure 3.1: Packet format.

A flit is the basic unit for flow control. It can be as large as a packet or as small as the physical channel width (also called phit). In this design, we set the flit size equal to the phit size, which is set as 24 bits. Figure 3.2 shows the flit format. The two bits at the front of each flit tell if the flit is the beginning flit of the packet (BOP) or the tail flit of the packet (EOP).

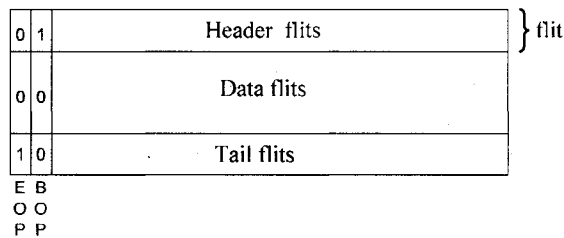
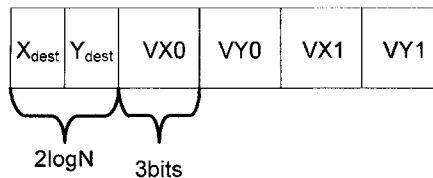


Figure 3.2: Flit format.

The header flit carries the same information of the header bits of a packet. The header flit format is shown in Figure 3.3, which includes the coordinates of the destination node's address ( $X_{dest}$ ,  $Y_{dest}$ ), and the routing vector generated by the routing algorithm (refer to Chapter 2.3 for the two routing algorithms). For a  $N \times N$  PRDT(2, 1) network, the destination field needs  $2\log N$  bits. The routing vector field stores the values of  $vecX_0$ ,  $vecY_0$ ,  $vecX_1$ ,  $vecY_1$ , each with 3 bits.





### 3.2 PRDT Router Design

The router designed is a Verilog HDL soft-core based on a library of parameterizable pre-designed blocks. It has up to 9 communication ports compliant with the links in a PRDT network, named N, NE, E, SE, S, SW, W, NW and L. The Local (L) is reserved to attach an IP, and the other ones (whose names follow the eight directions) are used for connecting other routers in eight directions. Each port has both input channel and output channel, e.g. NEin and NEout are the two channels for the port NE. Fig. 3.4 shows the nine ports and their directions.

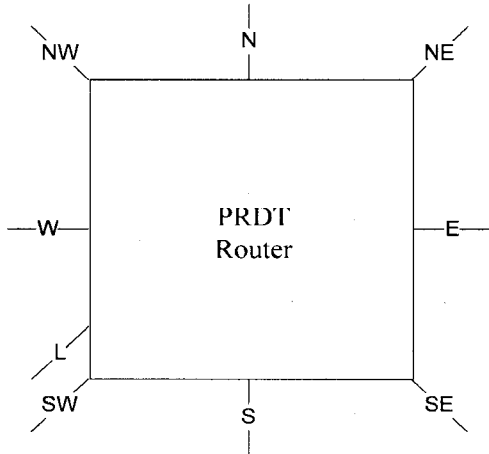


Figure 3.4: Communication ports of the router.

Before describing the building blocks of the router it is necessary to analyze the basic functions of the router. Figure 3.5 shows the function flow diagram of the router. The main function of the router includes buffering, routing, scheduling, switching, and flow control. When a packet is sent out in flits by a core or IP to a router port, it is stored in a buffer temporarily and waited to be forwarded. Buffering can be done at both the input and output sides of the router. The header flit is then read and an appropriate routing

algorithm is applied to decide the output port. Flow control is needed to synchronize the data transmission between routers and inside the router. The input channel module is designed to implement buffering, routing, and input flow control.

Scheduling is needed to solve the conflict when multiple flits from different input ports are destined to the same output port. A scheduler is designed to implement the scheduling scheme. A request signal is sent from the input controller to the scheduler, which does the arbitration and communicate with the requested output channel and sends the grant back to input controller.

After scheduling, then the flits are sent from the input ports to the output ports through a switching matrix. To implement switching, a crossbar switch is designed, which receives the control signal (Sel) from the scheduler.

At the output port, the flits are sent out to the output channel. An output channel module is designed to implement the flow control. It also notifies the scheduler whenever the output channel is empty or occupied with output grant (OG).

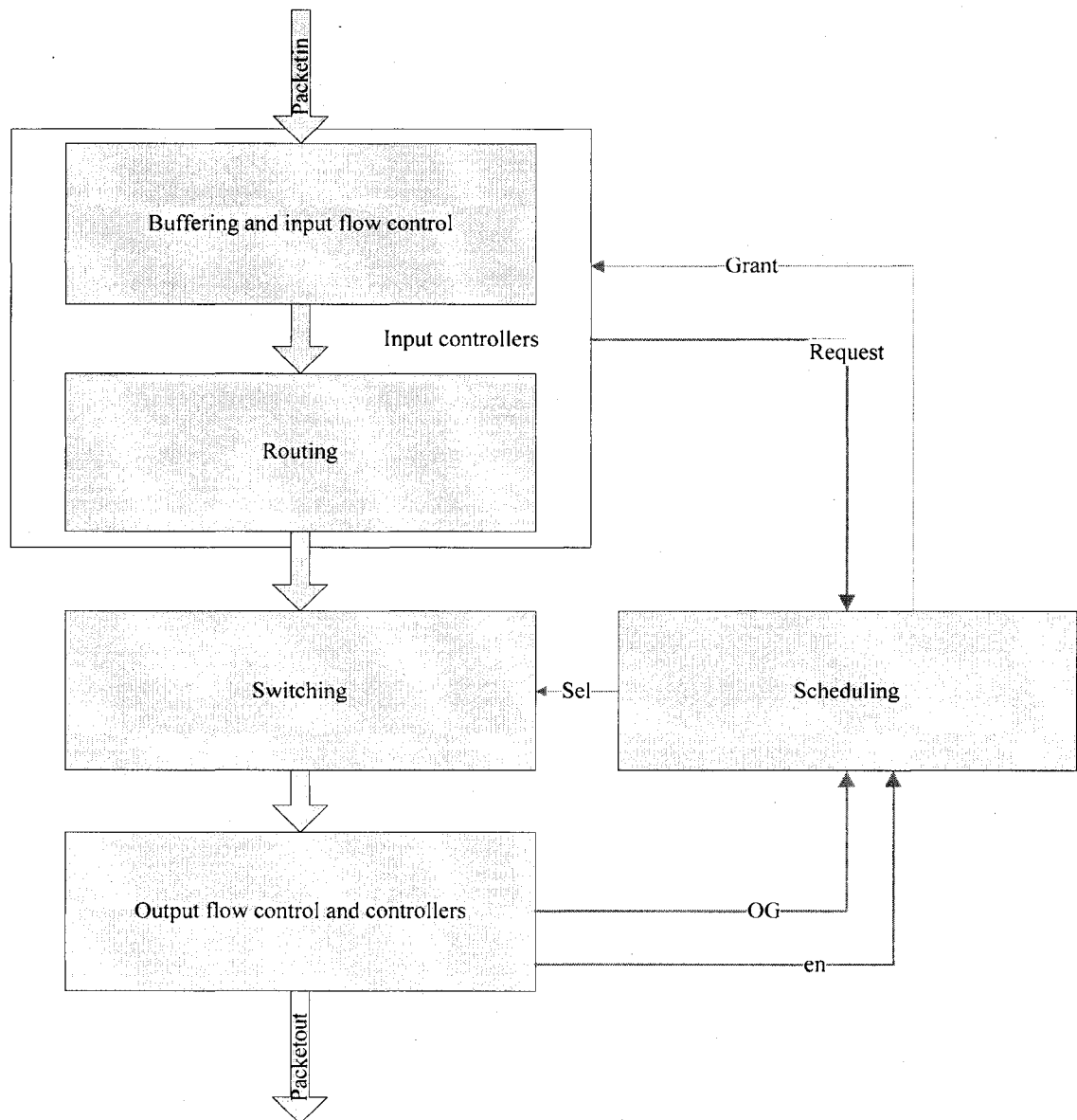


Figure 3.5: Function flow diagram of router.

### 3.3 Building Blocks of Router

The building blocks of the router include input channel module (ICM), output channel module (OCM), crossbar switch, and Scheduler. Fig. 3.6 shows the block diagram of the PRDT router, which includes up to 9 pairs of Input-Output controllers. Each building block is described in the following subsections.

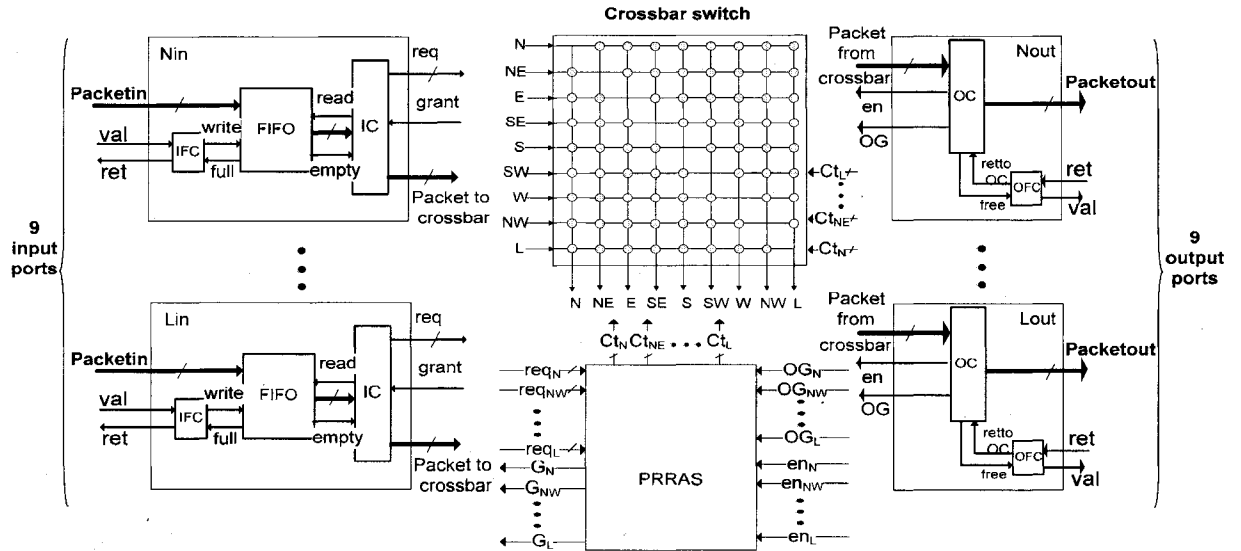


Figure 3.6: Block diagram of PRDT router.

### 3.3.1 Input Channel Module (ICM)

The ICM performs the following functions:

- (i) Buffers the incoming flits into the FIFO buffer and analyzes the flit at the head of the buffer.
- (ii) Performs the routing algorithm.
- (iii) Generates a request (req) to the scheduler for the appropriate output channel.

The ICM is shown in Figure 3.7. It is composed of three architectural blocks named Input Flow Controller (IFC), FIFO, and Input Controller (IC).

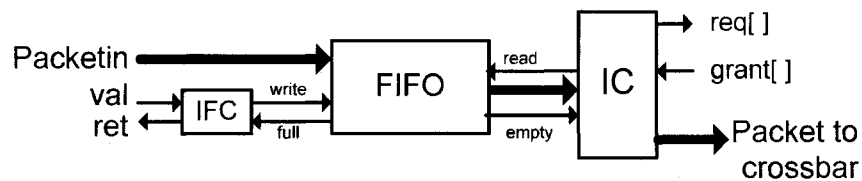


Figure 3.7: Input controller.

### 3.3.1.1 Input Flow Controller (IFC)

The IFC has two inputs (val, full) and two outputs (ret, write). The “full” signal comes from the FIFO and it toggles to one or zero when FIFO is full or not. The “val” signal, generated from the OFC of a requesting neighbor router, requests the permission to send the flit. The “write” signal sent to FIFO tells to write the data into the FIFO or not. The meaning of the “ret” signal varies with the flow control approach used. For the handshake one, it means that an acknowledgment for a flit received on the channel. For the credit-based approach, it means that a position was freed on the input buffer (i.e., the FIFO) of the receiver, and a credit is being returned to the sender. The sender can just send a flit if the receiver’s buffer is not full. The state of the buffer is monitored by using an up-down counter that is initialized at power-up with a number of credits equaling the receiver buffer depth. The counter is decremented when a flit is received and incremented when a credit is returned. If the credit counter equals 0, it means that the receiver’s buffer is full.

### 3.3.1.2 FIFO

FIFO is responsible to store flits of incoming packets before they are forwarded to an output channel. FIFO has three inputs (Packetin, write, read) and three outputs (dataout, full, empty). The “write” signal is from the IFC and the “read” signal is from IC to write the data and to read from FIFO, respectively. The “full” and “empty” signals basically inform the status of the buffer to IFC and IC.

### 3.3.1.3 Input Controller (IC)

The IC block performs the routing function. It detects the header flit received from FIFO, analyses the address field, runs the routing algorithm to select an output channel.

Two types of routing algorithms are implemented: the vector XY routing and the circular coded vector routing. In both the algorithms, the routing vector ( $vecX_1$ ,  $vecY_1$ ,  $vecX_0$ ,  $vecY_0$ ) is generated. As discussed in Chapter 2, the direction of the output channel is decided by the first variable with non-zero value and the variable's value will be updated accordingly. Figure 3.8 illustrates the output channel directions associated with the rank's directions. For example, if the routing vector has  $vecX_1=2$  and  $vecY_0=-1$ , it means the packet should be first routed in the SE direction for two steps and then routed in the N direction for one step. After the direction is selected, the IC emits a request to the scheduler for the destined output channel. Notice that the request from the IC of the ICM of one direction can only target to one of the other eight directions. For instance, i.e., the L channel cannot send a request to itself. Once it receives the grant from the scheduler, the header flit is updated with the new routing vector accordingly ( $vecX_1=1$  for the previous example) and sent to the output channel through a crossbar switch.

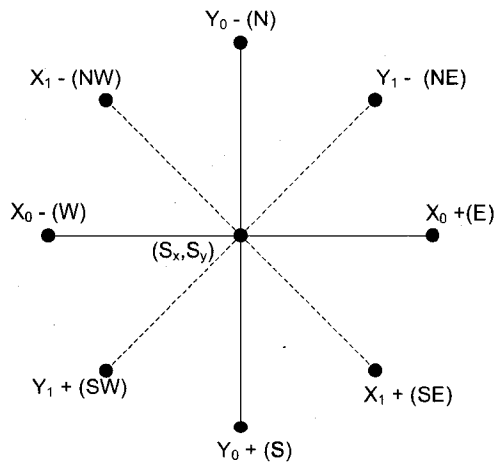


Figure 3.8: Directions of the output channels.

It is worthy to point out that only at the ICM for the local input port (the port that is connected to the core or IP), the routing algorithm is implemented and the routing vector is calculated. For other ports, the routing vector is just updated and stored back to the header flit.

There are three inputs (din, grant, empty) and three outputs (Packet to crossbar, req, read) in IC block. The “read” signal indicates FIFO data read is about to start. The flit is read from FIFO through the “din” input. Other signals are self-explainable.

### 3.3.2 Output Channel Module (OCM)

This module works independently of input controller or scheduler for the same router. As it works independently, the OG signal for the scheduler is readily available, even if the IC is not requesting for that output, thereby making the Scheduler more efficient and faster.

The output channel module is shown in Figure 3.9. It is composed of two architectural blocks named Output Flow Controller (OFC), and Output Controller (OC).

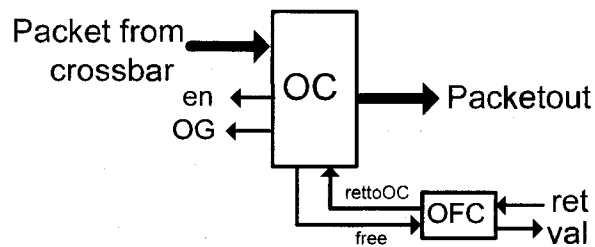


Figure 3.9: Output channel module.

#### 3.3.2.1 Output Flow Control (OFC):

The OFC block have two inputs (free, ret) and two outputs (rettoOC, val). OFC checks whether the OC is busy or not and sends a “val” (validate) signal to the IFC of the

requested neighbor router ports. After receiving the “val” signal, the IFC sends an appropriate ret (return) signal as described earlier. The “ret” signal is just passed onto the OC through the “rettoOC” output.

### 3.3.2.2 Output Controller (OC):

The OC block has two inputs (Packetfromcrossbar, rettoOC) and four outputs (OG, en, free, Packetout). OC updates its status (occupied or free) to the scheduler with the appropriate grant signal (OG). Once it gets the confirmation (“ret” signal) from the requested neighbor router, the packet is passed to “Packetout” output to the next router. en signal is given to the scheduler and used to control the input to encoder\_ctrl, which in turn control the mux-based crosspoint array. en signal is set when the eop reaches OC.

### 3.3.3 Crossbar Switch

System designers can construct non-blocking crossbar switch matrices by using crosspoint switching fabric integrated circuits, mostly built using advanced very large scale integration (VLSI) technology. Closing the switch at the appropriate crosspoint in the matrix creates a connection between an input and an output. An  $N \times M$  crossbar consists of  $N$  parallel horizontal wires (input) and  $M$  parallel vertical wires (output). Each horizontal wire crosses every vertical wire and a crossing switch may be placed at the cross-point. A crossing switch can be programmed to connect or disconnect the orthogonal wires at the cross-point. A crossbar is full if there is a crossing switch at each crosspoint; otherwise it is partial. Figure 3.10 shows an example of a partial crossbar where a diamond shape at a cross-point indicates a crossing switch. A full  $N \times M$  crossbar can route any set of  $k$  input signals to any set of  $k$  outputs in any permutation



provided  $k \leq N \leq M$  [17]. In particular, a full  $N \times N$  crossbar can do all permutations of  $N$  inputs/outputs.

Crossbars can be fabricated by using pass transistors or transmission gates as crossing switches with all horizontal wires in one metal layer and vertical wires in another. However, when  $N$  is large, a full  $N \times N$  crossbar is too expensive in area cost because the silicon area cost of switch modules is mainly attributed by the transistors used for switches and controls of the switches. Therefore, the study on crossbar design for programmable on-chip networks focuses on partial crossbars satisfying certain routing specifications, for example, Lemieux and Lewis [44] gives a several highly routable sparse crossbar designs.

For the XY routing, only the 72 out of 81 connections represented by circles are allowed (Figure 3.10). For instance, it is forbidden for an input channel of a given communication port to request the output channel of the same port. To allow fault-tolerant routing, it is allowable to have the input channels on the  $Y$  direction ( $N_{in}$  and  $S_{in}$ ) to request the output channels on the  $X$  direction ( $E_{out}$  and  $W_{out}$ ). Hence, the building cost of the crossbar switch is expected to be reduced.

For verilog soft-core implementation, multiplexers and splitters are used to implement the crossbar switch. Nine 8-to-1 multiplexers were used and is shown in the Figure 3.11. Shaded line represents a splitter. The control signals are provided by the scheduler.

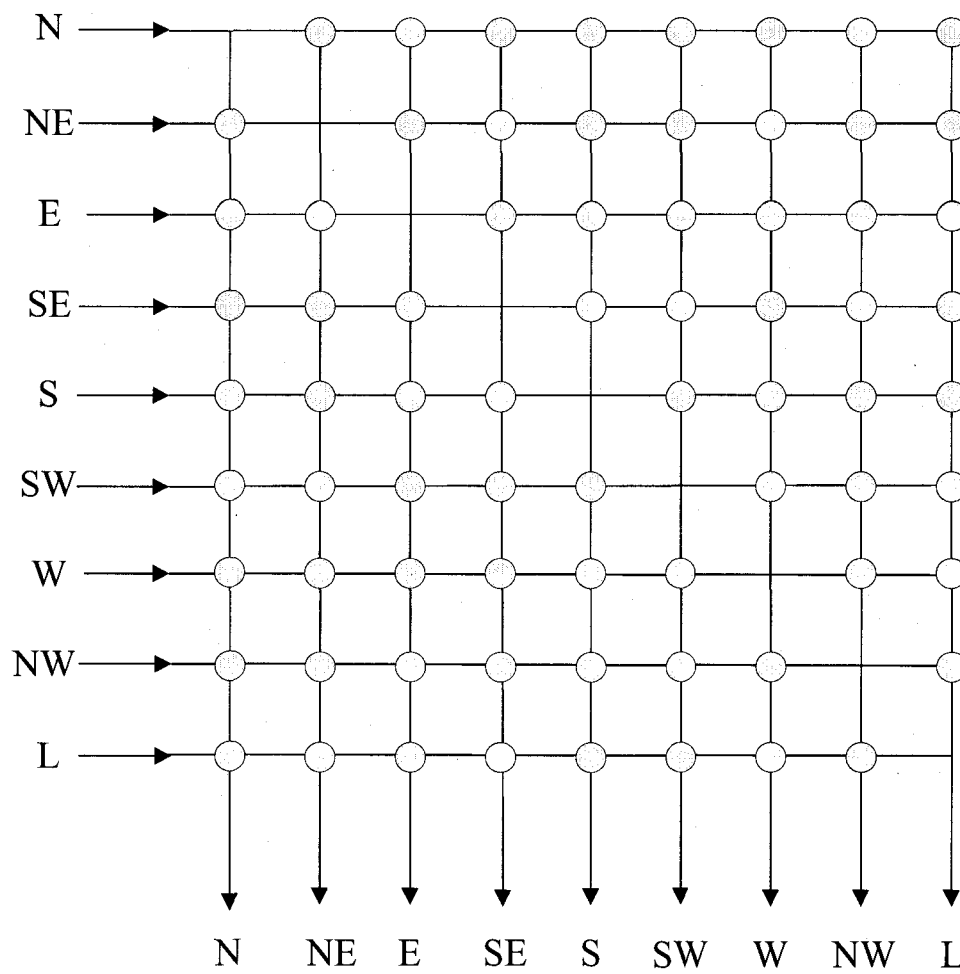


Figure 3.10: Crossbar structure.

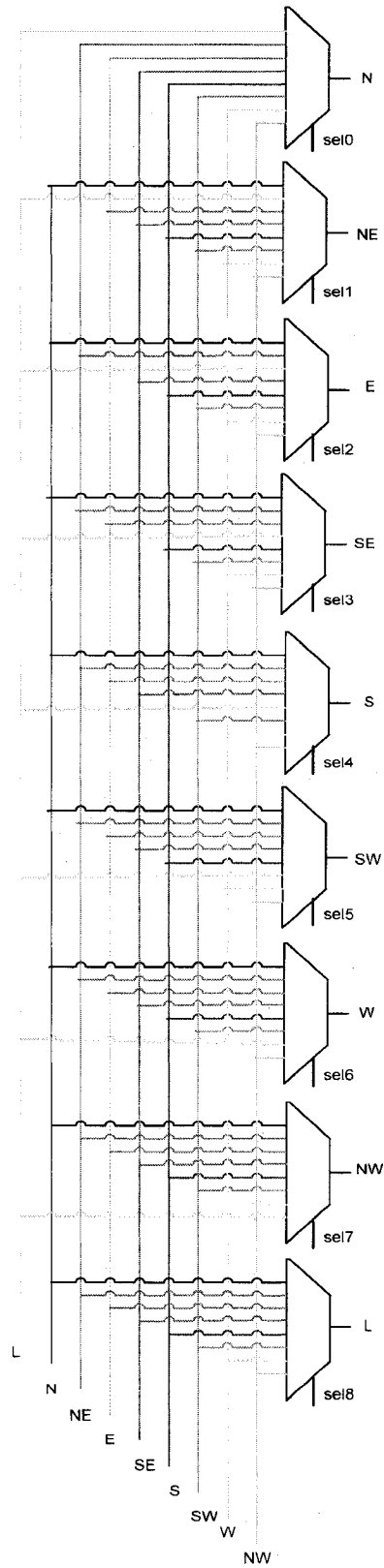


Figure 3.11: Mux-based crossbar.

### 3.3.4 Parameterized Round Robin Arbiter based Scheduler (PRRAS)

As a basic building block of a router, a fast and fair scheduler is critical to the efficiency of the router. In the ParIS router design [82], a distributed scheduling scheme is used, where an arbiter is located in each ICM and OCM. However, the distributed approach is not scalable with the router size increasing. Another problem is that the wiring cost (as each pair of ICM and OCM needs a wire for request signal and a wire for grant signal) and the delay involved in scheduling is not desirable. To achieve better scalability and reduce the wire cost and timing delay, the PRRAS adopts a centralized structure.

The major component of PRRAS is the Parallel Round Robin Arbiter (PRRA) [84][85], which is associated with one output port and responsible for arbitrating the requests to the output port from up to  $N$  input ports. The PRRA design is based on a simple binary tree structure, which makes it scalable for large  $N$ . In [84], it showed that the PRRA design achieves significant improvement in area and timing compared with other round-robin arbiter designs, such as the switch arbiter (SA) and the programmable priority encoder (PPE). In the next section, we will introduce the structure of PRRAS followed by the design of each component.

## 3.4 Design of PRRAS

As discussed in Sections 3.2 and 3.3, the PRRAS receives the requests from input ports and the OG status from output ports, and decides a matching between input ports and output ports. The grant signals will be sent back to the input ports. The PRRAS design is parameterized with the router size. Fig. 3.12 shows basic block diagram of

PRRAS for an  $N \times N$  router. The PRRAS has  $N$  request ( $R$ ) inputs, each one from an input port, and  $N$  OG and en inputs, each from an output port,  $N$  grant ( $G$ ) outputs, each to an input port, and  $N$  control ( $C$ ) outputs to the crossbar switch.

Figure 3.12 shows a block diagram of PRRAS and Figure 3.13 shows the complete design of the PRRAS, which consists of  $N$  PRRAs,  $N$  decoders, and  $N$  OR gates. A decoder is used to decode the request signal from an input port which gives the index of the output port that the input port requests to. The inputs of each PRRAS are provided by the outputs of the decoders in the following way; the first input comes from the first decoder, the second input comes from the second decoder, and so on. It is important to notice that each PRRAS may receive multiple simultaneous requests.

Since each input port only requests one output port, the grant to each input port can only come from one output. Hence, an OR gate is used to generate the grant signal to an input port. Similarly, the inputs to each OR gate are provided by the outputs of PRRAs as follows: the first input to all the OR gates comes from the first PRRAS, the second input to all the OR comes from the second PRRAS, and so on.

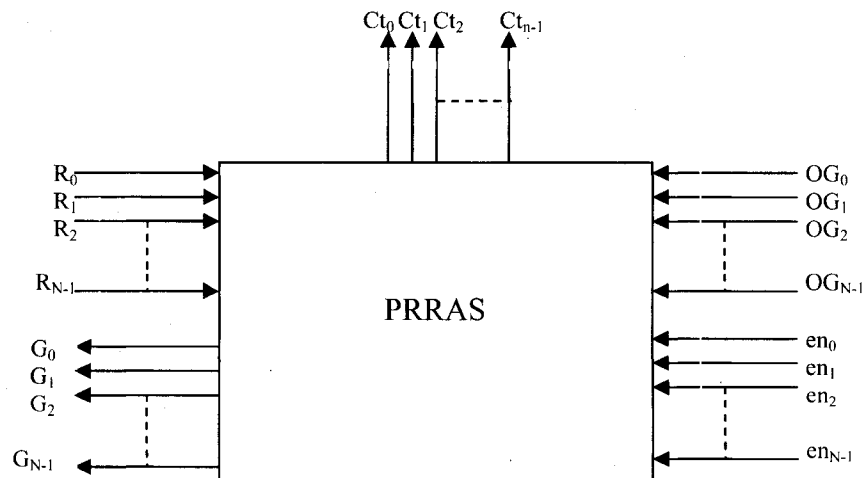


Figure 3.12: Block diagram of PRRAS.

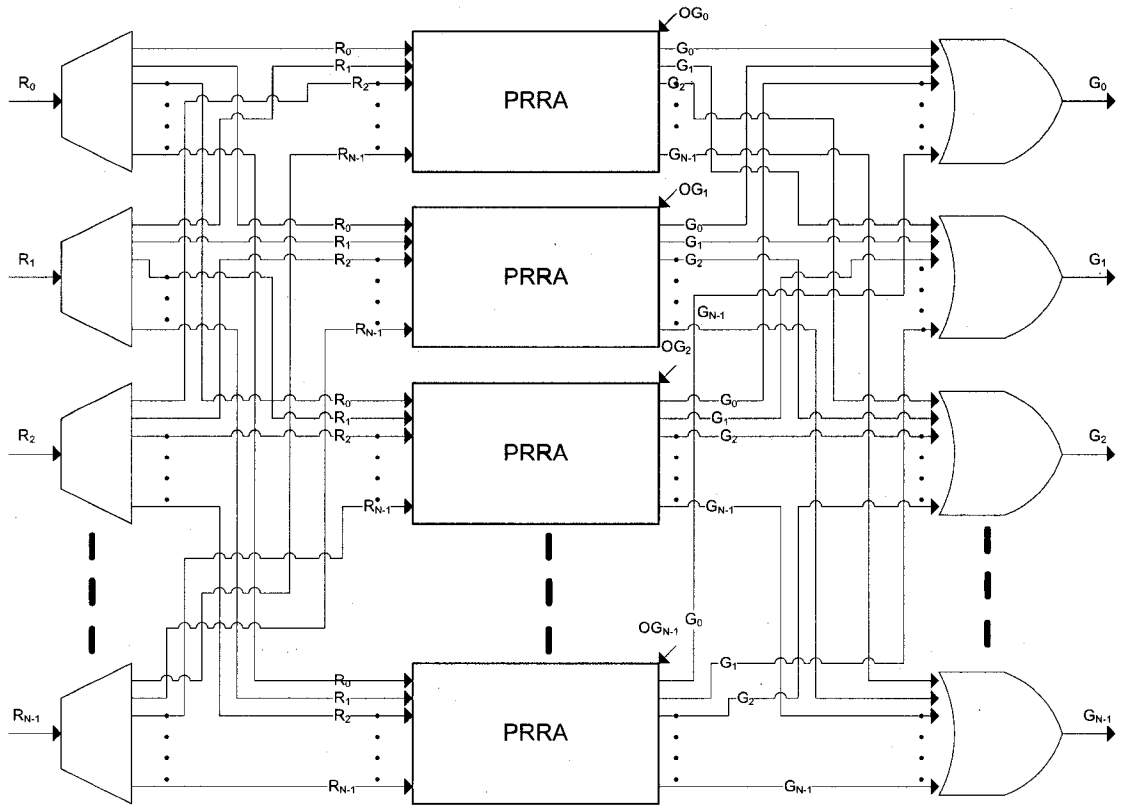


Figure 3.13: Complete Design of PRRAS.

In the following, the design of the decoder and PRRA will be discussed.

#### 3.4.1 Decoder

The input to each decoder is provided by the request signal from each input. As discussed before, the request signal is obtained from the routing vector. For a  $N \times N$  router, the request signal has  $\log_2 N$  bits.

A normal  $m$ -to- $2^m$  decoder may be used in the PRRAS design. Since the size of the router  $N$  may not equal to  $2^m$ , to save the wire cost, a  $\log_2 N$ -to- $N$  decoder is designed here. The decoder is parameterized with  $N$  as a parameter.

### 3.4.2 PRRA

PRRAS has two input signals to each arbiter viz. OG and en. The en and OG is generated from the output controller of the router. Only if the output controller is free, OG signal is sent and the arbitration is done.

There is an encoder\_ctrl in PRRAS, which is not shown in the figure 3.13, which provides the control signals for mux-based cross-point array through Ct, depending on the grants generated by the arbiters. The input to the encoder\_ctrl depends on the en signal generated by the OCM. en signal is used to check the eop of the packet, once the eop reaches the OCM, en signal will be set and then the grants from the arbiters will control the encoder\_ctrl accordingly. Thereby computation time and delay is reduced.

The arbiter design follows the PRRA design [84], which is reviewed as follows. The function of a PRRA is: Given binary inputs  $R_i$  and  $H_i$ ,  $0 \leq i \leq N-1$ , where  $R_i=1$  indicates a request from input  $i$  and  $H_i=1$  indicates the selection starts from  $R_i$ , compute binary grant outputs  $G_i$ ,  $0 \leq i \leq N-1$ . It is assumed that there is at most one  $H_i=1$ .

The following guidelines are used in PRRA design:

- (1) Use a tree to carry out the processing steps, such that the state information is collected in the up-trace (i.e. from leaves to the root), and the search is performed in the down trace;
- (2) Use combinational circuits as much as possible to fasten the design and the circuits must be simplified as much as possible; and
- (3) Use flip-flops to keep the current circular pointer information (which can be initialized as  $H_0=1$ ), and use the tree and grant signals to update flip-flops.

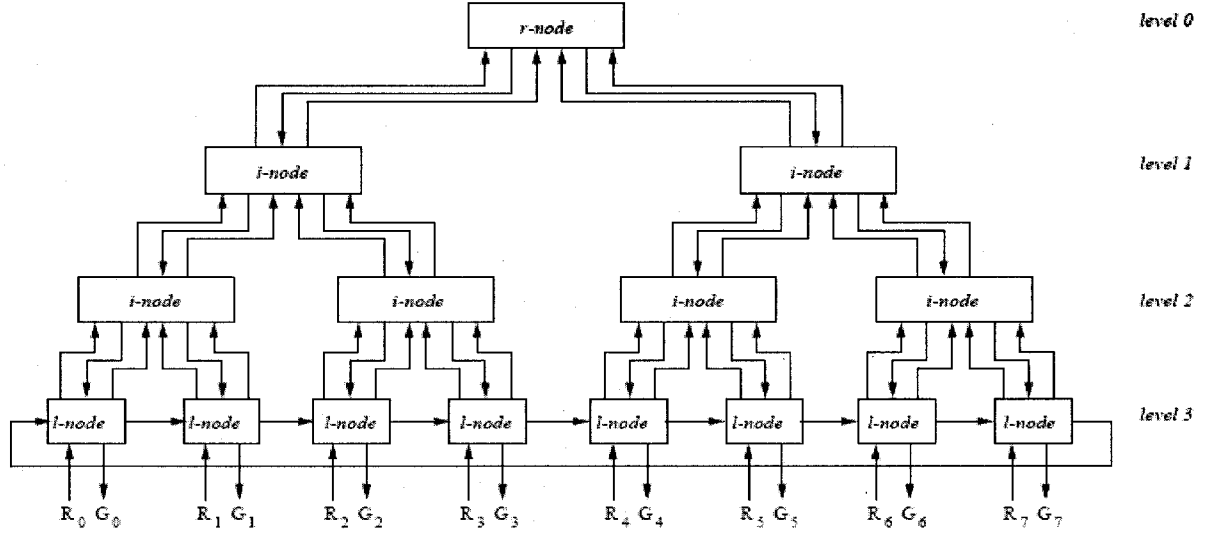


Figure 3.14: Structure of PRRA.

The basic idea of the PRRA design is to directly implement the PRRA-tree using hardware. Figure 3.14 shows the structure of a PRRA with 8 requests and its inputs and outputs. The RRA designed consists of three main components: 1). *i*-node, 2). *r*-node, 3). *l*-node. In the figure, there is no memory at the *r*-node and *i*-node to store the state information. Memories are only needed for storing the circular pointer at the *l*-node level. *l*-nodes are connected as a ring.

#### 3.4.2.1 *i*-node

An *i*-node is implemented as combinational circuit. It has four inputs from its two child nodes (which are either *l*-nodes or *i*-nodes):  $S_L^1$  and  $S_L^0$  from its left child, and  $S_R^1$  and  $S_R^0$  from its right child. It provides two outputs  $S^1$  and  $S^0$  to its parent node. If an *i*-node is the left (respectively, right) child of its parent node, then its  $S^1$  and  $S^0$  are identified as  $S_L^1$  and  $S_L^0$  ( $S_R^1$  and  $S_R^0$ , respectively) of its parent respectively. An *i*-node has one input  $G$  from its parent node. If this *i*-node is the left (resp. right) child node of its



parent node, this input is the  $G_L$  (resp.,  $G_R$ ) output of its parent node. It has two outputs  $G_L$  and  $G_R$  to its child nodes, which in turn are  $G$  inputs of its left and right child node respectively. The input and output relations of an  $i$ -node are specified by the following Boolean functions.

$$\begin{aligned} S^0 &= S_R^0 + S_L^0 \cdot \overline{S_R^1} \\ S^1 &= S_L^1 + S_R^1 \\ G_L &= G_{in} \cdot (S_L^0 \cdot \overline{S_R^0} + S_L^0 \cdot \overline{S_R^1} + S_L^1 \cdot \overline{S_R^0}) \\ G_R &= G_{in} \cdot (\overline{S_L^1} \cdot \overline{S_L^0} + \overline{S_L^0} \cdot S_R^0 + S_R^1 \cdot S_R^0) \end{aligned}$$

#### 3.4.2.2 $r$ -node

The implementation of  $r$ -node is same as the  $i$ -node except its one input ( $G$ ) is fed from OG of OCM and no  $S^1, S^2$  outputs. The Boolean functions are shown below:

$$\begin{aligned} G_L &= OG \cdot (S_L^0 \cdot \overline{S_R^0} + S_L^0 \cdot \overline{S_R^1} + S_L^1 \cdot \overline{S_R^0}) \\ G_R &= OG \cdot (\overline{S_L^1} \cdot \overline{S_L^0} + \overline{S_L^0} \cdot S_R^0 + S_R^1 \cdot S_R^0) \end{aligned}$$

#### 3.4.2.3 $l$ -node

Memories are only needed for storing the circular pointer at the  $l$ -node level. The entire processing is partitioned into two phases, up-trace for generating  $S^1, S^0$ , and down-trace for searching the desired  $l$ -node and generating grant signals. The state information  $S^1, S^0$  for all nodes is computed on-the fly recursively from  $l$ -nodes towards the  $r$ -node. Then, the partial grants ( $G_L$  and  $G_R$  signals) are generated from the  $r$ -node towards  $l$ -nodes in parallel by the same circuits. The circular pointer is updated according to the final grant after an arbitration cycle. Figure 3.15 shows how  $l$ -nodes are connected. Each dashed rectangle represents an  $l$ -node, which mainly consists of an RS flip-flop Head.

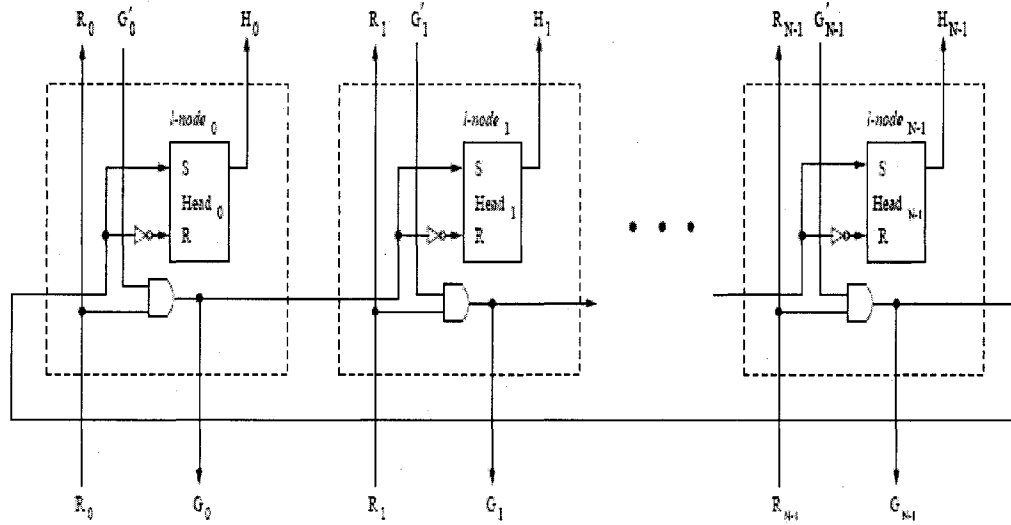


Figure 3.15: Structure of  $l$ -node.

### 3.4.2.3 PRRA Implementation

Rather than using a recursive approach as reported in [84], in this design, a parameterized PRRA is implemented. The PRRA design has three types of nodes,  $i$ -node,  $r$ -node, and  $l$ -node. The  $l$ -node design follows the design shown in Fig. 3.15. The  $r$ -node can be implemented using  $i$ -node by assigning the input  $G=1$ . For an  $N$ -input PRRA,  $N-1$   $i$ -nodes will be implemented. Actually as discussed before, the  $i$ -node design will not change with  $N$ . By this way, certain amount of area and timing can be saved compared with the previous PRRA design.

## CHAPTER 4

### OPTIMIZATION AND SIMULATION RESULTS

In this chapter, the simulation results of the router design are reported and discussed. The optimization of the scheduler is also discussed.

As specified in Chapter 3, the major components of the router are input channel module (ICM), output channel module (OCM) crossbar switch, and the scheduler. Two routing algorithms, the Vector Routing (VR) algorithm and the Circular Coded Vector Routing (CCVR) algorithm, are implemented in the ICM. Both are designed with fault tolerance of single fault. For each component, Verilog HDL code [8] [31] is generated and synthesized on Synopsys's design analyzer [68] using TSMC 0.18 $\mu$ m technology. Performance evaluation of each component and the whole router in terms of timing, area, and power consumption is conducted.

#### 4.1 Results of components of the router

##### 4.1.1 Input Channel Module

The ICM includes IC, FIFO and IFC. The FIFO is designed to contain 4 flits, which can be adjusted according to the real setting. Two types of input controllers (ICs) are designed: IC for the local port ( $IC_L$ ), the port that connects the local node to the router, and IC at other ports ( $IC_O$ ). The  $IC_L$  performs one of the routing algorithms, whereas  $IC_O$  simply does routing vector checking and update. Table 4.1 lists the area (in terms of

number of 2-input NAND gates), the timing delay, and power (both dynamic power and leakage power) of the IC<sub>L</sub> for two routing algorithms. The IC<sub>L</sub> using CCVR has larger area and consumes more power as since the computation in the CCVR algorithm involves more variables (implemented as registers) than the other one. However, IC<sub>L</sub> using CCVR has less timing delay.

Table 4.1: Results of area, timing, and power consumption of two types of IC<sub>L</sub>s.

		IC <sub>L</sub> Using Vector Routing (Fault Tolerance)	IC <sub>L</sub> Using Circular Coded Vector Routing (Fault Tolerance)
Total Area (2-input NAND gate)		242.64	316.215
POWER	Dynamic Power ( <i>mW</i> )	7.0567	11.1655
	Leakage Power ( <i>nW</i> )	27.3093	31.3723
Timing Delay ( <i>ns</i> )		11.73	7.23

Table 4.2: Results of area, timing, and power consumption of FIFO and IFC.

		FIFO	IFC
Total Area (2-input NAND gate)		274.0383	3.043
POWER	Dynamic Power ( <i>mW</i> )	4.8497	78.3008
	Leakage Power ( <i>nW</i> )	54.0854	71.5297
Timing Delay ( <i>ns</i> )		2.74	0.40

Table 4.3: Results of area, timing, and power consumption of ICMs with IC<sub>L</sub>.

		ICM Using Vector Fouting (Fault Tolerance)	ICM Using Circular Coded Vector Routing (Fault Tolerance)
Total Area (2-input NAND gate)		511.012	584.586
POWER	Dynamic Power ( <i>mW</i> )	6.0938	6.5496
	Leakage Power ( <i>nW</i> )	81.4663	85.5293
Timing Delay ( <i>ns</i> )		12.27	8.10

Table 4.2 lists the area, timing delay, and power of the FIFO and IFC, which do not have difference for two routing algorithms. Table 4.3 shows the results for ICMs for local ports, which are consistent with the results in Table 4.2, i.e., the ICM using CCVR has higher area cost but lower timing delay than the ICM using VR.

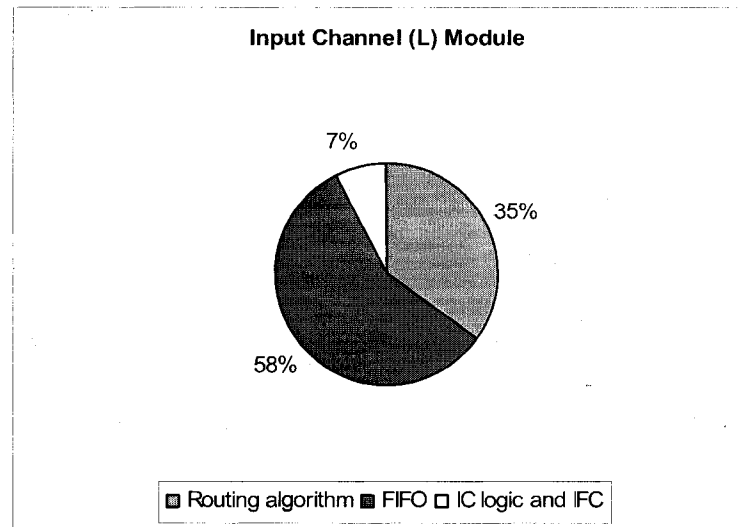


Figure 4.1: Area distribution of components in ICM<sub>L</sub>.

The pie chart (figure 4.1) shows the area distribution of ICM. It is evident that FIFO takes up a lot of space (58%) and it is unavoidable. 42% of the ICM is occupied by the IC<sub>L</sub>, in which 35% is occupied by routing algorithm module.

For the ICM with  $IC_O$ , considerable amount of area and timing can be saved due to the simple function performed at the  $IC_O$ . The results of the ICMs with  $IC_O$  are shown in the Table 4.4. Compared with the ICM with  $IC_O$ , 35% more saving is achieved in area than the ICM with  $IC_L$ . Since the operation of two routing algorithm do not differ much for  $IC_O$ , the results for the two ICMs are very close.

Table 4.4: Results of area, timing, and power consumption of ICMs with  $IC_O$ .

		ICM Using Vector Routing (Fault Tolerance)	ICM Using Circular Coded Vector Routing (Fault Tolerance)
Total Area (2-input NAND gate)		339.9488	339.1928
POWER	Dynamic Power ( $mW$ )	5.669	5.6829
	Leakage Power ( $nW$ )	60.0262	60.5103
Timing Delay ( $ns$ )		2.74	2.74

#### 4.1.2 Output Channel Module

The output channel module performs very little function as explained in Chapter 3 and the OCM design is independent of the routing algorithm. The results of the OCM are shown in Table 4.5.

Table 4.5: Results of area, timing, and power consumption of OCM.

		OCM
Total Area (2-input NAND gate)		37.089
Power	Dynamic Power ( $mW$ )	1.0456
	Leakage Power ( $nW$ )	7.5771
Timing Delay (fs)		0.33

#### 4.1.3 Scheduler

PRRA is the main component of the Parameterized Round Robin Arbiter Based Scheduler (PRRAS) design. The area and timing results of the PRRA design will dominate the results of the scheduler design. In this section, simulation results of new PRRA design and previous PRRA design [84] [85] are analyzed using Synopsys' design analyzer.

Both the previous PRRA design and the new PRRA design are modeled using verilog HDL codes and synthesized on Synopsys using TSMC  $0.18\mu m$  technology. Both designs were optimized under the same operating conditions and the tool is directed to optimize area cost of each design. Table 4.6 and Table 4.7 show the area (in terms of number of 2-input NAND gates) and timing results ( $ns$ ) of both PRRA design for arbiter input size  $N = 4, 8, 16$ . Although the results depend on the standard cell library used, they represent the relative performance of these designs.

Table 4.6: Area results (in number sq microns) of two PRRA designs.

Arbiter Size	New PRRA Design	Previous PRRA Design
4	558.72	604.80
8	1365.12	1411.20
16	2977.92	3024.00

Table 4.7: Timing results(*ns*) of two PRRA designs.

Arbiter Size	New PRRA Design	Previous PRRA Design
4	0.92	0.91
8	1.45	1.44
16	1.97	1.96

It is clear from the above tables that the new PRRA design reduces a considerable amount of area compared with the previous PRRA design. The area improvement of the new PRRA design over the previous PRRA design is 7.6% when  $N=4$  and the improvement effect is less significant when  $N$  is increasing. Timing result shows that the new PRRA design increases the timing ( $0.01ns$ ) compared with the previous design, which is negligible. Similarly power consumptions between the two designs are the same. Hence, the new PRRA design is adopted in the PRRAS design.

In the PRRAS design, the number of PRRAs is determined by the number of inputs to the router ( $N$ ). And the number of inputs to a PRRA must be a 2's power which is equal or greater than  $N$ . For the PRDT router design, the router has 9 inputs, hence, there are 9 16-input PRRAs in the PRRAS. In this case, 7 inputs in the each arbiter are wasted.

In order to solve the problem, the scheduler is optimized in the following way. According to the XY routing principle, an input port will not request to the output port on the same direction of itself. For example, the NE input port will not request for the NE output port. Therefore, each input port will request the remaining 8 output ports in the router. The PRRA arbiter associated with each output port accepts the 8 inputs which represent the requests from the possible 8 input ports. As such, the scheduler is reduced to have 9 8-input PRRAs. Table 4.8 shows the results of the non-optimized scheduler and



optimized scheduler. One can see that the area and power of the optimized scheduler is reduced by 50%.

Table 4.8: Results of area, timing, and power consumption of PRRAS with 9 inputs.

		Non-optimized	Optimized
Total Area (2-input NAND gate)		1638.654	1151.50
Power	Dynamic Power ( <i>mW</i> )	23.4471	22.3977
	Leakage Power ( <i>nW</i> )	152.3767	82.3323
Timing Delay ( <i>ns</i> )		4.81	4.23

#### 4.2 Results of Complete Router

The PRDT router with 9 input/output ports is built using all the components described earlier. The area distribution of the router using the non-optimized designs is shown in Fig. 4.2. The crossbar switch and the scheduler are the two components that occupy the most part of the area of the router. The optimization of the scheduler (as described in Section 4.1.3) and the crossbar switch (will be described in Chapter 5) has been conducted. Table 4.9 summarizes the results of the router design based on the vector routing (VR) algorithm and the router design based on the circular coded vector routing algorithm (CCVR) with and without optimization. Fig. 4.3 and Fig. 4.4 illustrate the results in bar graph. . It is clear that the router design using CCVR consumes a little more area (around 1%) but has significant less timing delay (around 27%) than the router design using VR. The power results of the two designs are very close.

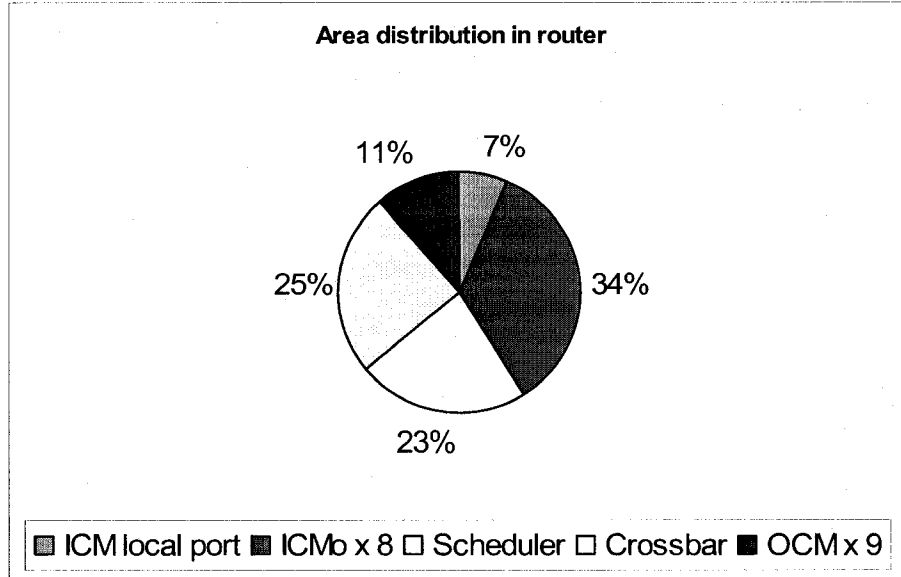


Figure 4.2: Area Distribution of All Components of the Router.

Table 4.9: Results of area, timing, and power consumption of two routers with and without optimization.

	Non-Optimized Router Using VR	Non-Optimized Router Using CCVR	Optimized Router Using VR	Optimized Router Using CCVR
Area (2-input NAND gates)	7058.07	7131.82	6192.84	6266.415
Dynamic Power ( <i>mW</i> )	67.2197	67.0591	61.3938	62.1609
Leakage Power ( <i>nW</i> )	802.5729	807.1178	754.9756	759.0386
Timing Delay ( <i>ns</i> )	25.25	18.41	12.27	8.10

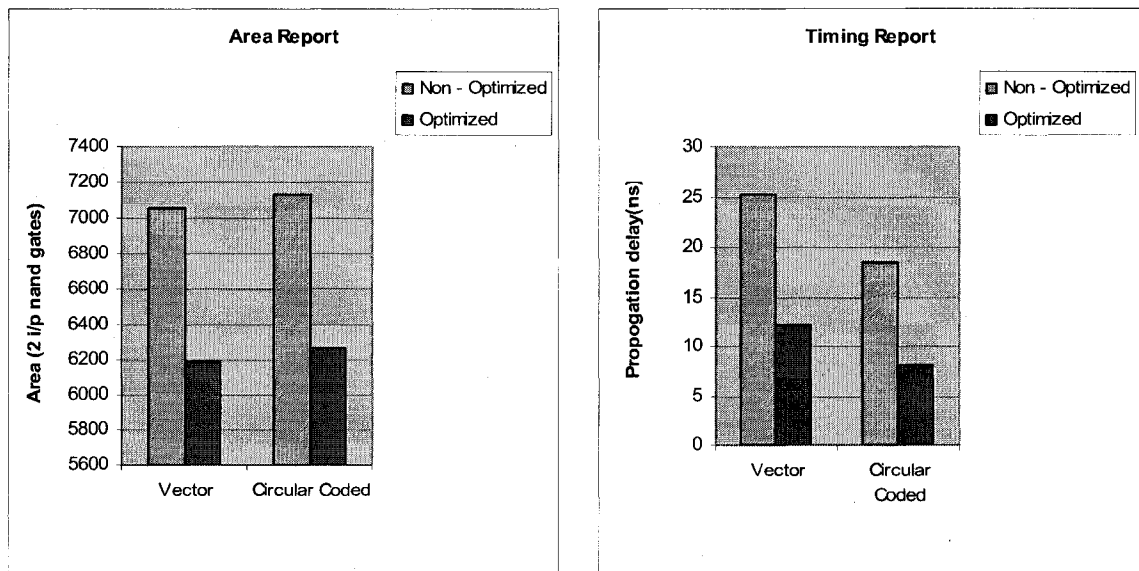


Figure 4.3: Area and Timing Results between Two Routers (Normal and Optimized).

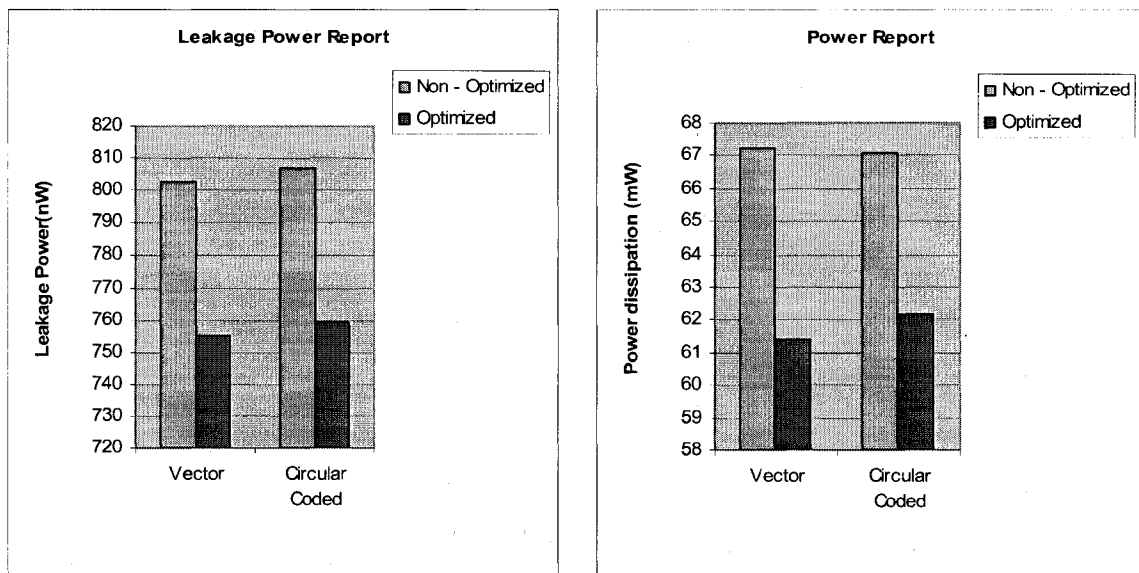


Figure 4.4: Comparison of Power Results of Two Routers (Normal and Optimized).

#### 4.3 Report on PRDT Network

The optimized router design is used to build the PRDT network. The PRDT networks with 4x4 routers and 8x8 routers have been built. Both designs have been generated in

verilog HDL code and synthesized on Synopsys's design analyzer using TSMC 0.18 $\mu$ m technology. The synthesis of 8x8 PRDT network design is not successful due to the limit of resources available in the design library. In the following, we report the results of 4x4 PRDT network.

As shown in Fig. 2.1(b), 4x4 PRDT(2, 1) is a special network that does not need all the nine ports. Three variations of the network design are considered, 1) using 6-port routers instead of 9-port routers, 2) using 9-port routers but with necessary connections (as shown in Fig. 2.1 (b)), 3) using 9-port routers and with complete connections (as shown in Fig. 2.1 (a)). Table 4.10 and Table 4.11 list the area, power, and timing results for the 4x4 PRDT networks constructed with the routers using VR algorithm and the routers using CCVR algorithm, respectively. In both versions, compared with the design with 9-port routers with minimum connections, the design with 6-port routers significantly reduces the area consumption (around 38%), timing delay (around 8%), and power consumption (around 31% in dynamic power). There is not much difference between the results of the design using 9-port routers with minimum connections and the design using 9-port routers with complete connections.

Table 4.10: Results of 4x4 PRDT Network Using Routers with CCVR Algorithm.

	Area (2 i/p NAND gates)	Timing Delay ( <i>ns</i> )	Dynamic Power ( <i>mW</i> )	Leakage Power ( <i><math>\mu</math>W</i> )
6-port routers	60334.8	15.64	357.2202	7.7023
9-port routers (minimum connections)	98398.9867	16.94	518.4472	11.6425
9-port routers (complete connections)	98916.6733	16.94	518.5639	11.6425

Table 4.11: Results of 4x4 PRDT Network Using Routers with VR Algorithm.

	Area (2 i/p nand gates)	Timing Delay ( <i>ns</i> )	Dynamic Power ( <i>mW</i> )	Leakage Power ( <i>uW</i> )
6-port routers	59115.42	21.36	357.2134	7.6090
9 port routers (minimum connections)	97179.3867	23.40	518.4091	11.5491
9 port routers (completes)	97697.08	23.40	518.5259	11.5491

Comparing the corresponding results of Table 4.10 and Table 4.11, the design using CCVR-based routers has a little worse area result but better timing result than the design using VR-based routers, which is consistent with the trend shown in Table 4.9.

#### 4.4 Timing Analysis of the 4x4 PRDT network:

The computational time of the network mainly depend on  $t_h$ , the time for header to reach the destination.

$$t_h = t_r * \text{number of hops}$$

where,  $t_r$  = header flit propagation time per hop.

From the simulation results  $t_r$  and  $t_h$  is calculated. Fig. 4.5 shows a graph comparing  $t_h$  of the 4 x4 PRDT network build with CCVR-based router and VR-based router (keeping the source node = (0,0)). It is evident from the graph  $\max(t_h)$  is 12 clock cycles for VR-based network and 9 clock cycles for CCVR based network.  $t_r$  is found out from the simulation.

Max (no. of hops for VR based 4x4 PRDT network) = 3.

Max (no. of hops for CCVR based 4x4 PRDT network) = 2.

From the results it is proved that in PRDT 4x4 network, Circular Coded Vector routing algorithm takes the minimum number of hops and hence the shortest path compared to Vector Routing algorithm.

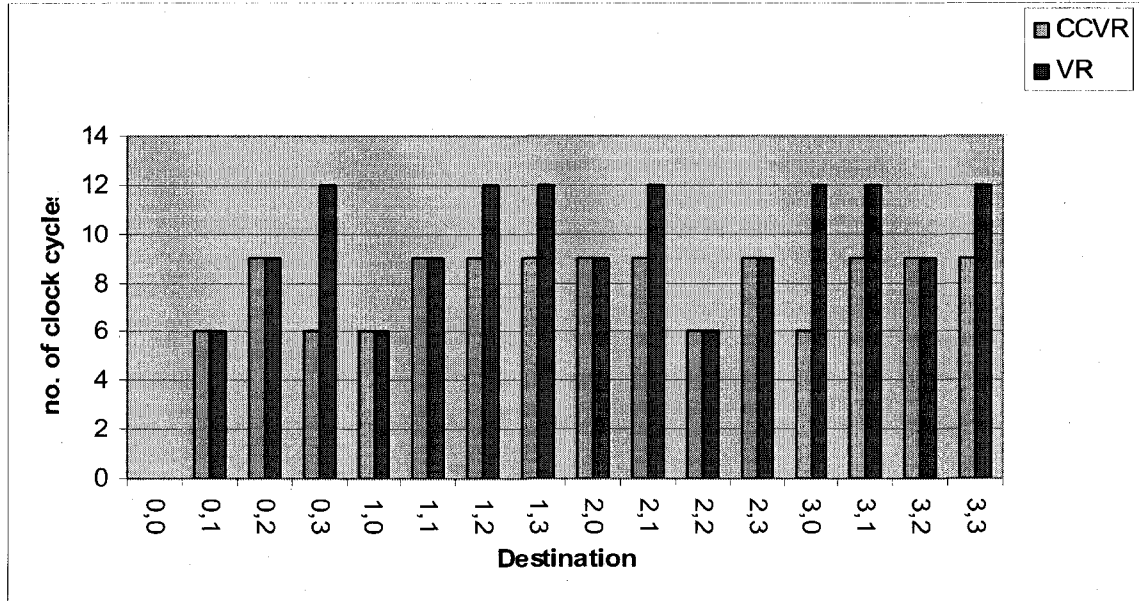


Figure 4.5: Calculation of  $t_h$  w.r.t (0,0) as source.

## CHAPTER 5

### STUDY OF CROSSBAR SWITCH DESIGN

From Figure 4.2, it is evident that the crossbar switch design takes above 25% area of the router. Lots of research [43] [17] is going in this field to improve the design of the crossbar. There are different ways to design a crossbar, and the tradeoffs involved in those designs are explained in [56]. This chapter gives a comparative study of the crossbar switch designs based on crosspoints and multiplexers.

#### 5.1 Overview of Two Types of Crossbar Switch Design

There are two basic methods for implementing crossbar switch in hardware. Some customized ASICs and standard products rely on an  $N$ -way multiplexer at each output port to select data input from the input ports. This method is referred as MUX-based design in the following text. Many semiconductor vendors have built crossbar switch products based on this simple methodology. Unfortunately, this type of design is limited in terms of architectural flexibility and performance, and is difficult to be implemented efficiently at transistor level.

The second method implements a cross-point array that has a crossing switch at each intersection of the input wire and the output wire. This method is referred as crosspoint-

based This method offers greater flexibility than the other method, making it feasible to build larger size crossbar switches economically using modern VLSI technology.

For both methods, both the verilog HDL code and the manual design using MAGIC layout editor are conducted and compared. Functionally a crosspoint-switch can come in two versions: bit or bus. The bit-version switch can only switch one bit from an input to an output while the bus-version can switch the multiple data bits (decided by the bus bitwidth) from an input to an output. The bus-version switch design is coded in verilog HDL. The bit-version switch design is carried out using MAGIC layout editor.

## 5.2 Designs in Verilog Code

The MUX-based design is explained in the chapter three. Generally, the number of inputs to a multiplexer is equal to the number of inputs of the crossbar switch. For 9-port PRDT router, the crossbar switch should have 9 inputs and 9 outputs. Hence it requires 9 9-input multiplexers. However, using the XY routing principle as explained in Chapter 3, each output only needs consider 8 inputs. By this way, 9 8-input multiplexers are needed in this design. The MUX-based design is implemented in verilog HDL and synthesized using Synopsys' design analyzer. Table 5.1 lists the design with and with this optimization. The optimized MUX-based design considerably reduces the area, timing delay, and power consumption compared with the non-optimized design.



Table 5.1: Results of MUX-based 9x9 crossbar switch design.

		Non-optimized	Optimized
Total Area (2-input NAND gate)		1789.704	1631.734
Power	Dynamic Power ( $mW$ )	23.9764	19.7669
	Leakage Power ( $nW$ )	45.3580	42.7494
Timing Delay ( $ns$ )		1.86	1.64

The crosspoint-based design is also implemented using verilog HDL code, synthesized. Table 5.2 shows the area and timing results of 4x4 crosspoint-based design and 4x4 MUX-based design. The crosspoint-based design is much more expensive than the MUX-based design in terms of area cost. This is due to the fact that the optimization provided by the design tool is not sufficient for the design written in behavior description code. Hence, it is necessary to conduct the comparison of the manual design using MAGIC layout editor.

Table 5.2: Comparison of MUX-based and crosspoint-based crossbar designs (both in verilog code).

	4x4 Mux-based design	4x4 Switch-based design
Area (2 ip nand gates)	487.648	1070.868
Data Arrival Time(ns)	1.92	2.10

### 5.3 Design using Layout Editor

On MAGIC layout editor, the bit-version switch design is conducted. For  $N \times N$  crossbar switch, the MUX-based design uses  $N$  output multiplexers and  $N$  splitters as

shown in Fig. 3.11. Transmission gates are used to build the multiplexer [58]. Fig. 5.1 shows a two-input multiplexer design.

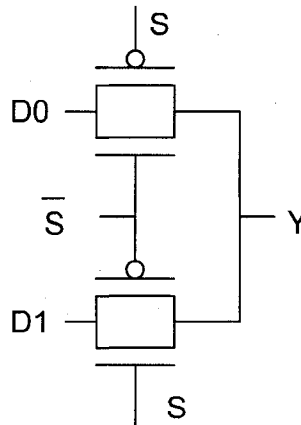


Figure 5.1: 2-Input multiplexer based on transmission gates.

For comparative study, transmission gates are also used to build crosspoint-based crossbar switch [36]. Fig. 5.2 shows the crossing switch designed with transmission gate and Fig. 5.3 shows how they are connected in a 4x4 crossbar switch.

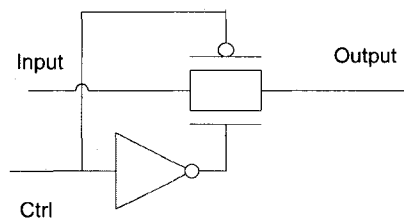


Figure 5.2: Crossing switch designed with transmission gate.

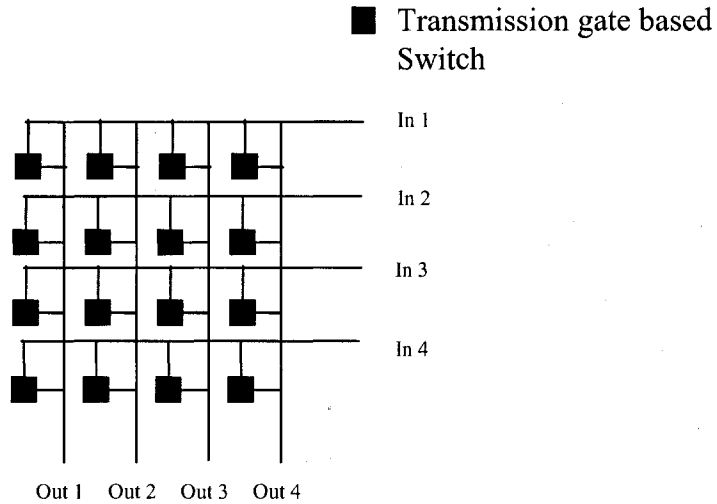


Figure 5.3: 4x4 Crossbar switch using crossing switches.

Both bit-version MUX-based and crosspoint-based designs are laid out using MAGIC layout editor for different crossbar switch sizes (2x2, 4x4, and 8x8 respectively). Area is directly measured from the layout by counting the number of grids. Power and timing analysis is done using Hspice. TSMC 0.18 $\mu$ m technology is used for Hspice simulation and the waveforms are viewed using Synopsys Awaves. Table 5.3 lists the results of area, timing, power consumption, and number of transistors used vs. crossbar switch size of two types of bit-version switch design. Figures 5.4-5.7 illustrate these results.

From Table 5.3 and the figures, one can see that the crosspoint-based design requires more area when N is 2 or 4 while less area when N is 8 than the MUX-based design. This is not consistent with the trend shown in the no. of transistors needed in these designs since the wiring cost counts a significant amount in the area result. The timing result and power result of the MUX-based design are worse than the crosspoint-based design.

Table 5.3: Report on crossbar (using layout)

Crossbar	Area (sq. microns)	Propagation Delay ( <i>ns</i> )	Total power ( $\mu W$ )	No. of Transistors
Crosspoint-based (2x2)	55.2825	0.01	1.53	16
Crosspoint-based (4x4)	215.9136	0.01	6.14	64
Crosspoint-based (8x8)	817.5006	0.03	24.9	256
MUX-based (2x2)	34.5384	0.01	1.53	12
MUX-based (4x4)	165.24	0.025	12.2	72
MUX-based (8x8)	860.8761	0.05	73.4	336

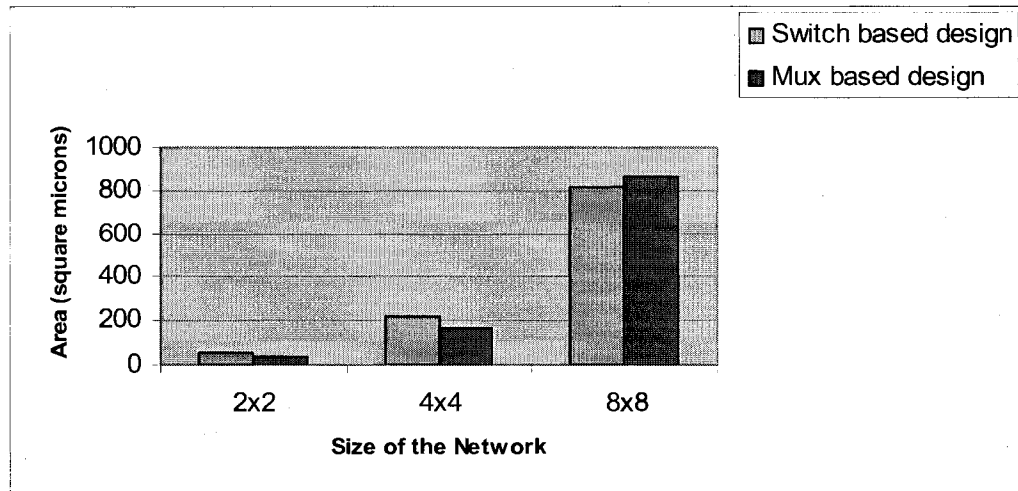


Figure 5.4: Area vs. Crossbar switch size.

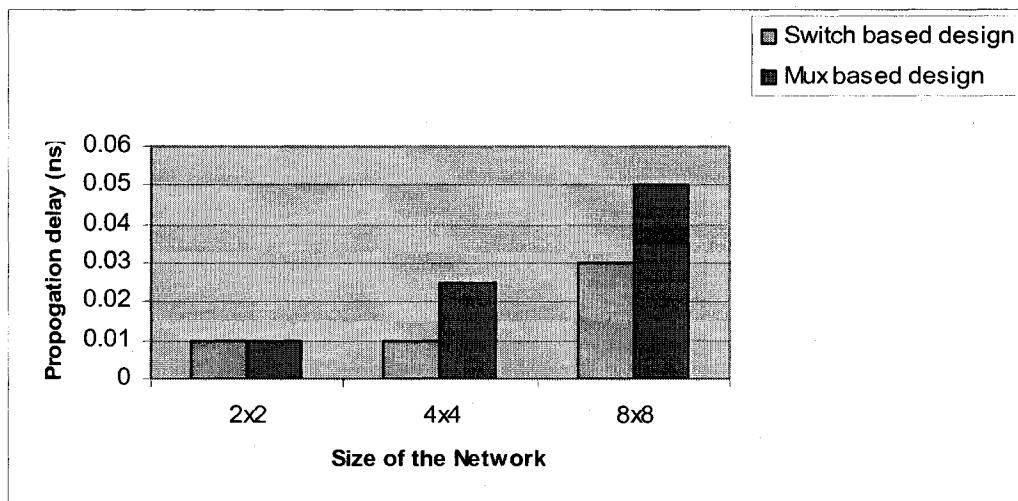


Figure 5.5: Timing delay vs. Crossbar switch size.

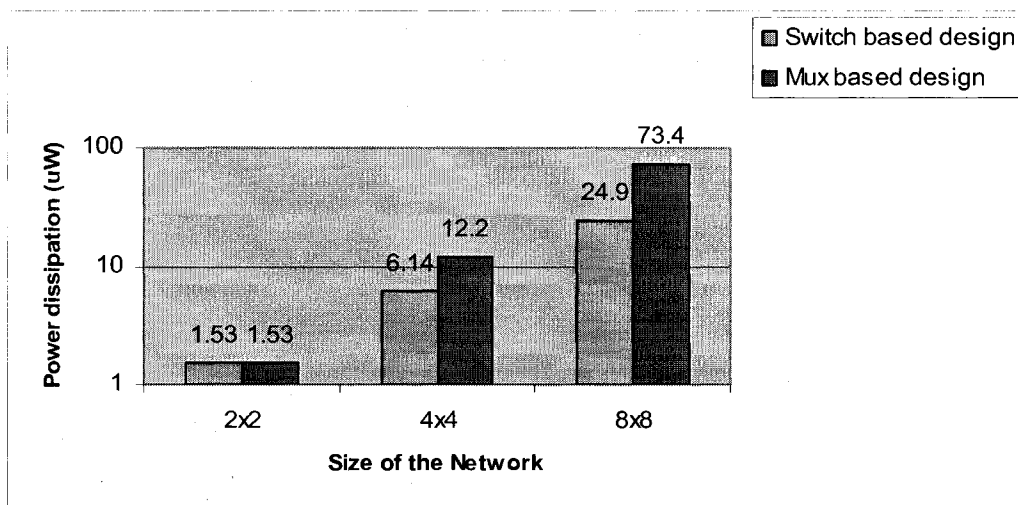


Figure 5.6: Power consumption vs. Crossbar switch size.

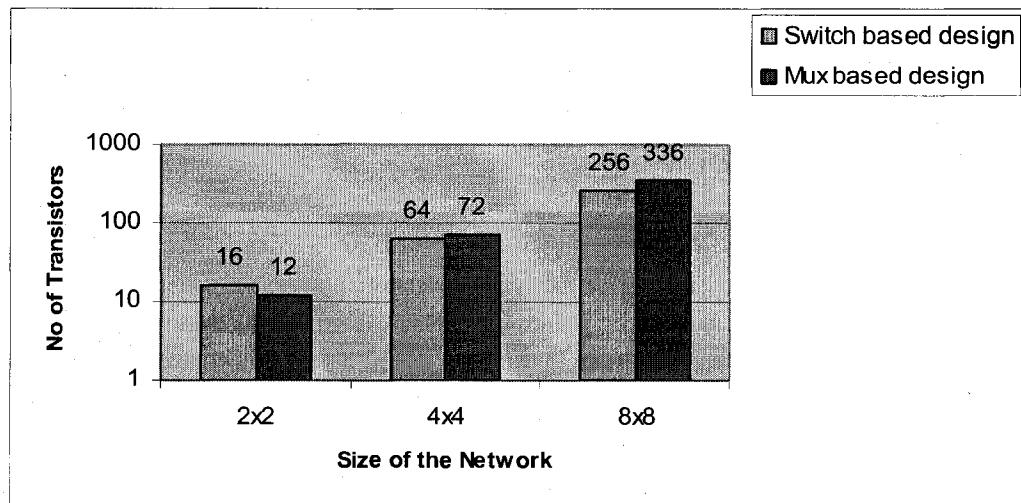


Figure 5.7: Transistor count vs. Crossbar switch size.

Through the comparison, the following summary is given for bit-version  $N \times N$  crossbar switch design:

1. Crosspoint-based design consumes less area than the MUX-based design.
2. Crosspoint-based design is symmetrical while the MUX-based design is not.
3. The wiring in MUX-based design is more complex than the crosspoint-based design.
4. Number of crossing switches in crosspoint-based design:  $N^2$ .
5. Number of multiplexers in MUX-based design:  $N$ .
6. The architecture of the crossing switch does not change with different switch size, whereas the architecture of the multiplexer changes with different switch size. For example, the 2x2 MUX uses 6 gates, the 4X4 MUX uses 18gates (equivalent to 3 2x2 MUX).
7. Number of transistors of crosspoint-based design:  $4 * N^2$

8. Number of transistors of MUX-based design:  $N * (n * (N-1))$ , where  $n$  is the number of transistors require to build a 2x2 MUX ( $n = 6$  in the design reported).

Hence, we expect that for larger  $N$ , the MUX-based design will consume more area and no. of transistors than the crosspont-based design.

Note that, the above results are for bit-version switch design only. From Table 5.3, an estimation of bit-version switch design can be derived. For a 24-bit crossbar switch, roughly 1536 bit-based design is calculated. For crosspoint-based design, the estimation is given as: area consumption = 200002.8 sq. microns, power consumption = 597  $\mu W$ , and no. of transistors = 9437184. For MUX-based design, it is estimated that area consumption = 246787.8 sq. micron (which is almost equal to area report generated using Synopsys design\_analyzer, see Table 5.1), no. of transistors = 14146560.

Though the crosspoint-based design is promising in terms of area, timing, and power consumption, it is difficult to integrate the design into the router design. Hence, in Chapter 4, we employ the MUX-based crossbar switch design.

## CHAPTER 6

### CONCLUSION AND FUTURE WORK

#### 6.1 Conclusion

In this thesis, a new router design has been conducted for the PRDT-based NoCs. The PRDT router designed is based on parameterized and synthesizable components coded in verilog HDL. These components include input controller module, output controller module, crossbar switch, and the scheduler. In the input controller module, two routing algorithms with fault tolerance capability are implemented, i.e., the vector routing algorithm and the circular coded vector routing algorithm. The synthesized results for all the components have been reported.

As the two major components of the router, the scheduler design and the crossbar switch design have been optimized. The design of the PRRAS uses the new PRRA design which improves the previous PRRA design. For crossbar switch, two design methods have been studied and compared in verilog HDL code and MAGIC layout editor, one using MUX-based design, the other using crosspoint-based design. Further, these designs are optimized according to the XY routing principle. Through this optimization, significant improvement has been achieved in terms of area, timing, and power consumption.

An important feature of this router design is that it can be simply modified to be used for mesh/torus-based network as the PRDT network naturally embeds the mesh/torus.



## 6.2 Future Work

Currently, the crossbar design is not parameterized since the MUX design is changing with the switch size. The study shows that the crosspoint-based design has better results than the MUX-based design. It is advisable to replace the current design by a better parameterized crossbar.

In current router design, the two routing algorithms implemented are deterministic. In the future, adaptive routing schemes can be implemented. In those algorithms, virtual channels [4] [46] can be added at the input controller module.

## REFERENCES

- [1] G. Ascia et.al, "Multi-objective mapping for mesh-based NoC architectures," in *Proc. CODES*, 2004, pp. 182-187.
- [2] L. Benini and G. De Micheli, "Networks on chip: a new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70-78, Jan. 2002.
- [3] T. Bjerregaard, J. Sparso, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Proc. DATE*, vol. 2, 2005, pp. 1226-1231.
- [4] T. Bjerregaard and J. Sparso, "Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip," in *Proc. IEEE Norchip Conference*, 2004, pp.269-272.
- [5] T. Bjerregaard, and S. Mahadevan, 2006. "A survey of research and practices of Network-on-chip," *ACM Comput. Surv.*, vol. 38, no.1, Jun. 2006.
- [6] J. T. Brassil, "Deflection routing in certain regular networks," *Ph.D. dissertation, Univ. California at San Diego*, 1991.
- [7] G.-M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Trans. Parallel and Distrib. Syst.*, vol. 11, no. 7, pp. 729-738, Jul. 2000
- [8] Michael D. Cilette, *Starter's Guide to Verilog 2001*, Prentice Hall, 2004.
- [9] W. J. Dally and C. L. Seitz. "The torus routing chip," *Distributed Computing*, vol. 1, no. 3, pp. 187-196, 1986.
- [10] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Trans. Comput.* vol. 39, no. 6, pp. 775-785, Jun. 1990.
- [11] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proc. DAC*, 2001, pp. 684-689.
- [12] X. Duan, Y. Yang, M. Yang, L. Li, Y. Jiang, "Topology and binary routing schemes of a PRDT-based NoC," in *Proc. ITNG*, 2007, pp. 920-924

- [13] A. V. De Mello, L. C. Ost, F. G. Moraes, and N. L. V. Calazans, "Evaluation of routing algorithms on mesh based NoCs," Faculdade de Informatica, PUCRS – Brazil, Tech. Rep. TR-040,, May 2004.
- [14] J. Dielissen, A. Radulescu, K. Goossens, and E. Rijpkema. "Concepts and implementation of the phillips network-on-chip," in *Proc. IPSOC*, 2003.
- [15] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann, San Francisco, 2003.
- [16] T. Dumitras and R. Marculescu, "On-chip stochastic communication," in *Proc. DATE*, 2003, pp. 790-795.
- [17] H. Fan, Yu-Liang Wu, "Crossbar based design schemes for switch boxes and programmable interconnection networks," in *Proc. Design Automation Conference*, vol. 2, 2005. pp. 910-915.
- [18] T. Felicijan and S. B. Furber. "An asynchronous on-chip network router with quality-of-service (QoS) support," in *Proc. IEEE Int'l SoC Conf.*, 2004, pp. 274-277.
- [19] M. Forsell, "A scalable high-performance computing solution for networks on chips," *IEEE Micro.*, vol. 22, no. 5, pp. 46-55, Sept. 2002.
- [20] C. J. Glass and L. M. Ni. "The turn model for adaptive routing," in *Proc. ISCA*, 1992, pp. 278 – 287.
- [21] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet switched interconnections," in *Proc. DATE*, 2000, pp. 250 -256.
- [22] A. Hemani, *et al.*, "Network on a chip: an architecture for billion transistor era," in *Proc. IEEE NorChip Conf.*, 2000.
- [23] M. Harmanci, , N. Escudero, , Y. Leblebici, , and P. Ienne, "Quantitative modelling and comparison of communication schemes to guarantee quality-of-service in networks-on-chip," in *Proc. Int'l Symp. Circuits and Systems (ISCAS)*, 2005, pp. 1782–1785.
- [24] J. Henkel, W. Wolf, and S. Chakradhar, "On-chip networks: a scalable, communication-centric embedded system design paradigm," in *Proc. 17th Int'l Conf. VLSI Design*, 2004, pp. 845-851.

- [25] T. Henriksson, D. Wiklund, D. Liu, "VLSI implementation of a switch for on-chip networks," in *Proc. Int'l Workshop DDECS*, Apr. 2003.
- [26] J. Hu and R. Marculescu, "DyAD-smart routing for networks-on-chip," *Proc. DAC*, Jun. 2004. pp. 260-263.
- [27] J. Hu and R. Marculescu, "Application-specific buffer space allocation for networks-on-chip router design," in *Proc. IEEE/ACM Int'l Conf. Computer Aided Design*, 2004, pp. 354-361.
- [28] J. Hu, R. Marculescu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Trans. Computer-Aided Design*, vol. 24, no. 4, 2005, pp. 551-562.
- [29] W. Hung, *et al.*, "Thermal-aware IP virtualization and placement for Networks-on-Chip architecture," in *Proc. ICCD*, 2004, pp. 430-437.
- [30] K. Hwang and Z. Xu, *Scalable Parallel computing: technology, architecture, programming*, New York: *McGraw-Hill*, 1997.
- [31] IEEE Standards Board, *IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language*, New York: IEEE Inc., 1996.
- [32] International Technology Roadmap for Semiconductors Roadmap, Available: <http://public.itrs.net/>.
- [33] M.F. Jacome, H.P. Peixoto, "A survey of digital design reuse," *IEEE Design & Test of Computers*, vol. 18, no. 3, pp. 98-107, May-June 2001.
- [34] A. Jantsch, J. Öberg, and H. Tenhunen, "Editor note for special issue on networks on chip," *Journal of Systems Architecture*, vol. 50, Feb. 2004.
- [35] A. Jalabert, S. Murali, L. Benini, and G.D. Micheli, "xpipesCompiler: a tool for instantiating application specific networks on chip," in *Proc. DATE*, 2004, pp. 18-31.
- [36] E. Juarez, L. Cominelli, and D. Mlynek, "VLSI for telecommunication systems," Available: <http://lsiwww.epfl.ch/LSI2001/teaching/webcourse/ch11/ch11.6.html>.
- [37] F. Karim, A. Nguyen, and S. Dey, "An interconnect architecture for networking systems on chips," *IEEE Micro*, pp. 36-45, Sept./Oct. 2002,

- [38] N. Kavaldjiev and G. M. Smit, "An energy-efficient network-on-chip for a heterogeneous tiled reconfigurable systems on-chip," in *Proc. Euromicro Symp. Digital System Design (DSD)*, 2004, pp. 492-498.
- [39] P. Kermani and L. Kleinrock. "Virtual cut-through: a new computer communication switching technique," in *Computer Networks*, vol. 3, pp. 267-286, Sept. 1979.
- [40] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C.R. Das, "A low latency router supporting adaptivity for on-chip interconnects," in *Proc. DAC*, 2005, pp. 559-564.
- [41] S. Kumar, *et al.*, "A network on chip architecture and design methodology," *Proc. IEEE Computer Society Symp. VLSI*, 2002, pp. 117-124.
- [42] M. Kreutz, *et al.*, "Communication architectures for system-on-chip," in *Proc. Symp. Integrated Circuits and Systems Design*, 2001, pp. 14-19.
- [43] K. Lee, S.-J. Lee, H.-J Yoo, "A high-speed and lightweight on-chip crossbar switch scheduler for on-chip interconnection networks," in *Proc. 29<sup>th</sup> European Solid-State Circuits Conf. (ESSCIRC)*, 2003, pp. 453-456.
- [44] G. Lemieux and D. Lewis, *Design of Interconnection Networks for Programmable Logic*, Boston: Kluwer-Academic Publisher, 2003.
- [45] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network-on-chip," in *Proc. DATE*, 2004, pp. 890-895.
- [46] R. Mullins, A. West and S. Moore, "Low-Latency Virtual Channel Routers for On-Chip Network," in *Proc. Int. Symp. Computer Architecture*, 2004, pp. 188-197.
- [47] S. Murali, G. De Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," in *Proc. DATE*, Feb. 2004, pp. 896-901.
- [48] S. Murali, G. De Micheli. "SUNMAP: a tool for automatic topology selection and generation for NoCs," in *Proc. DAC*, vol. 4, 2004.
- [49] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, vol. 26, pp. 62-76, Feb. 1993.

- [50] S. Mahadevan, M. Storgaard, and J. Madsen, "ARTS: a system-level framework for modeling MPSoC components and analysis of their causality," in *Proc. 13th Int'l Symp. Modeling, Analysis and Simulation of Computer and Tele. Syst. (MASCOTS)*, 2005, pp. 480-483.
- [51] C. Neeb, M. Thul, and N. Wehn, "Network-on-chip-centric approach to interleaving in high throughput channel decoders," in *Int'l Symp. Circuits and Systems (ISCAS)*, 2005. pp. 1766-1769.
- [52] E. Nilsson, M. Millberg, J. Oberg, and A. Jantsch, "Load distribution with the proximity congestion awareness in a network on chip," in *Proc. Design, Automation and Test in Europe Conf. and Exhibition*, 2003, pp. 1126-1127.
- [53] U.Y. Ogras, J. Hu, and R. Marculescu, "Key research problems in NoC design: a holistic perspective," in *Proc. CODES+ISSS*, 2005, pp. 69-74.
- [54] U. Y. Ogras and R. Marculescu. "Energy- and performance- driven customized architecture synthesis using a decomposition approach," in *Proc. DATE*, 2005.
- [55] P.P. Pande, *et al.*, "Design of a switch for network on chip applications," in *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS)*, 2003, pp. 217-220.
- [56] A. Pedler, "Approaching crosspoint switches," Available: <http://www.commsdesign.com/main/2000/08/0008top.htm>.
- [57] A. Pinto, *et al.*, "Efficient synthesis of networks on chip," in *Proc. ICCD*, 2003, pp. 146-150.
- [58] J. M Rabaey, *et al.*, *Digital Integrated Circuits*, 2nd Edition, Prentice-Hall, 2002.
- [59] E. Rijpkema, *et al.*, "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip," *IEE Proc. Computers and Digital Techniques*, vol. 150, no. 5, pp. 294-302, Sept. 2003.
- [60] D. Rostislav, V. Vishnyakov, E. Friedman, R. Ginosar, "An asynchronous router for multiple service levels networks on chip," in *Proc. Asynchronous Circuits and Systems*, 2005, pp. 44-53.
- [61] I. Saastamoinen, *et. al.* "Buffer implementation for Proteo Network-on-Chip". In *Proc. Intl. Symp. on Circuits and Systems*, vol. 2, May 2003, pp. 113-116.

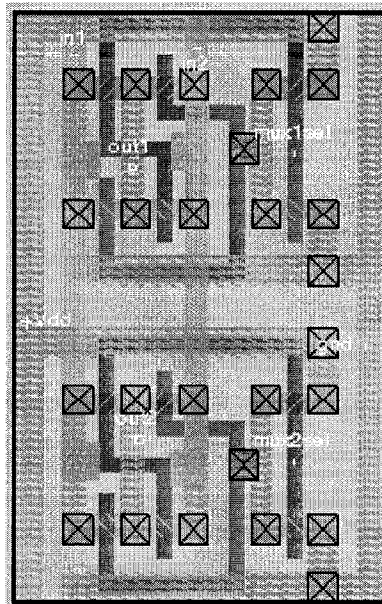
- [62] S. Sathe, D. Wiklund, D. Liu, "Design of a switching node (router) for on-chip networks," in *Proc. ASIC*, vol. 1, 2003, pp. 75-78.
- [63] L. Shang *et al.*, "PowerHerd: dynamically satisfying peak power constraints in interconnection networks," *IEEE Trans. Computer-Aided Design*, vol. 25, no. 1, pp. 92-110, Jan. 2006.
- [64] E.S. Shin, V.J. Mooney, and G.F. Riley, "Round-robin arbiter design and generation," in *Proc. Int. Symp. Sys. Syn. (ISSS)*, 2002, pp. 243-248.
- [65] K.G. Shin and S.W. Daniel, "Analysis and implementation of hybrid switching". *IEEE Trans. Computers*, vol. 45, no. 6, pp. 684 - 692, Jun. 1996.
- [66] K. Srinivasan, *et al.*, "Linear programming based techniques for synthesis of network-on-chip architectures," in *Proc. ICCD*, 2004, pp. 422-429.
- [67] H. Sullivan and T.R. Bashkow, "A large scale, homogeneous, fully distributed parallel machine," *Proc. 4th Int'l Symp. Computer Architecture*, 1977, pp. 105-117.
- [68] Synopsys                      Synthesis                      Tutorial,                      Available :  
[http://www.facweb.iitkgp.ernet.in/~apal/lpcs2007/webpages/271\\_Syn\\_tut.pdf](http://www.facweb.iitkgp.ernet.in/~apal/lpcs2007/webpages/271_Syn_tut.pdf).
- [69] H. Tanaka, *et al.*, *The Massively Parallel Processing System, JUMP-1*, Int'l Organizations Services Press, 1996.
- [70] T.T. Ye, L. Benini, and G. De Micheli, "Analysis of power consumption on switch fabrics in network routers," in *Proc. Design Automation Conference*, 2002, pp. 524-529.
- [71] T.T. Ye, L. Benini, G. De Micheli, "Packetization and routing analysis of on-chip multiprocessor networks," *Journal of Systems Architecture: the EUROMICRO Journal*, vol. 50, pp.81-104, Feb. 2004.
- [72] J. Wilinski, "An introduction to the MAGIC VLSI design layout system," Available: [http://www.glue.umd.edu/~newcomb/vlsi/magic\\_tut/Magic\\_x3.pdf](http://www.glue.umd.edu/~newcomb/vlsi/magic_tut/Magic_x3.pdf).
- [73] P.T. Wolkotte, G.J.M. Smit, G.K. Rauwerda, and L.T. Smit, "An energy-efficient reconfigurable circuit-switched network-on-chip," in *Proc. Parallel and Distributed Processing Symp.*, 2005, pp.155a - 155a.

- [74] J. Wu, "A deterministic fault-tolerant and deadlock-free routing protocol in 2-D meshes based on odd-even turn model," *Proc. 16th Int'l Conf. Supercomputing*, 2002, pp. 67-76.
- [75] G. Yang, M. Yang, Y. Yang, Y. Jiang, "On the physical layout of PRDT-based NoCs," in *Proc. ITNG*, 2007, pp. 729-33.
- [76] M. Yang, T. Li, Y. Jiang, Yulu Yang, "Fault-tolerant routing schemes in RDT(2,2,1)/ $\alpha$ -based interconnection network for networks-on-chip designs," in *Proc. Parallel Architectures, Algorithms and Networks (ISPAN)*, 2005, pp. 52-57.
- [77] Y. Yang, *et al.*, "Recursive diagonal torus: an interconnection network for massively parallel computers," *IEEE Trans. Parallel and Distrib. Syst.*, vol. 12, no. 7, pp. 701-715, Jul. 2001.
- [78] Y. Yang, H. Amano, H. Shibamura, and T. Sueyoshi, "Recursive diagonal torus: an interconnection network for massively parallel computers," in *Proc. 5<sup>th</sup> IEEE Symp. Parallel and Distrib. Processing*, 1993, pp. 591-594.
- [79] Y. Yu, T. Li, Xiaoshe Dong, Yulu Yang, "Fault-tolerant routing algorithm for RDT structure," in *Proc. ISPAN*, 2005, pp. 248-255.
- [80] Y. Yu, M. Yang, Y. Yang, and Y. Jiang, "A RDT-based interconnection network for scalable NoC designs," *Proc. ITCC*, 2005, pp. 723-728.
- [81] C.A. Zeferino, M.E. Kreutz, and A.A. Susin, "RASoC: a router soft-core for networks-on-chip," in *Proc. Design, Automation and Test in Europe Conf. and Exhibition*, vol. 3, 2004, pp. 198-203.
- [82] C.A. Zeferino, F.G.M.E. Santo, and A.A. Susin, "ParIS: a parameterizable interconnect switch for networks-on-chip," in *Proc. Integrated Circuits and Systems Design*, 2004, pp. 204-209.
- [83] C. A. Zeferino and A. A. Susin, "SoCIN: A Parametric and Scalable Network-on-Chip," in *Proc. SBCCI*, 2003, pp. 169-174.
- [84] S. Q. Zheng and M. Yang, "Algorithm-hardware codesign of fast parallel round-robin arbiters," *IEEE Trans. Parallel and Distrib. Syst.*, vol. 18, no. 1, Jan. 2007.

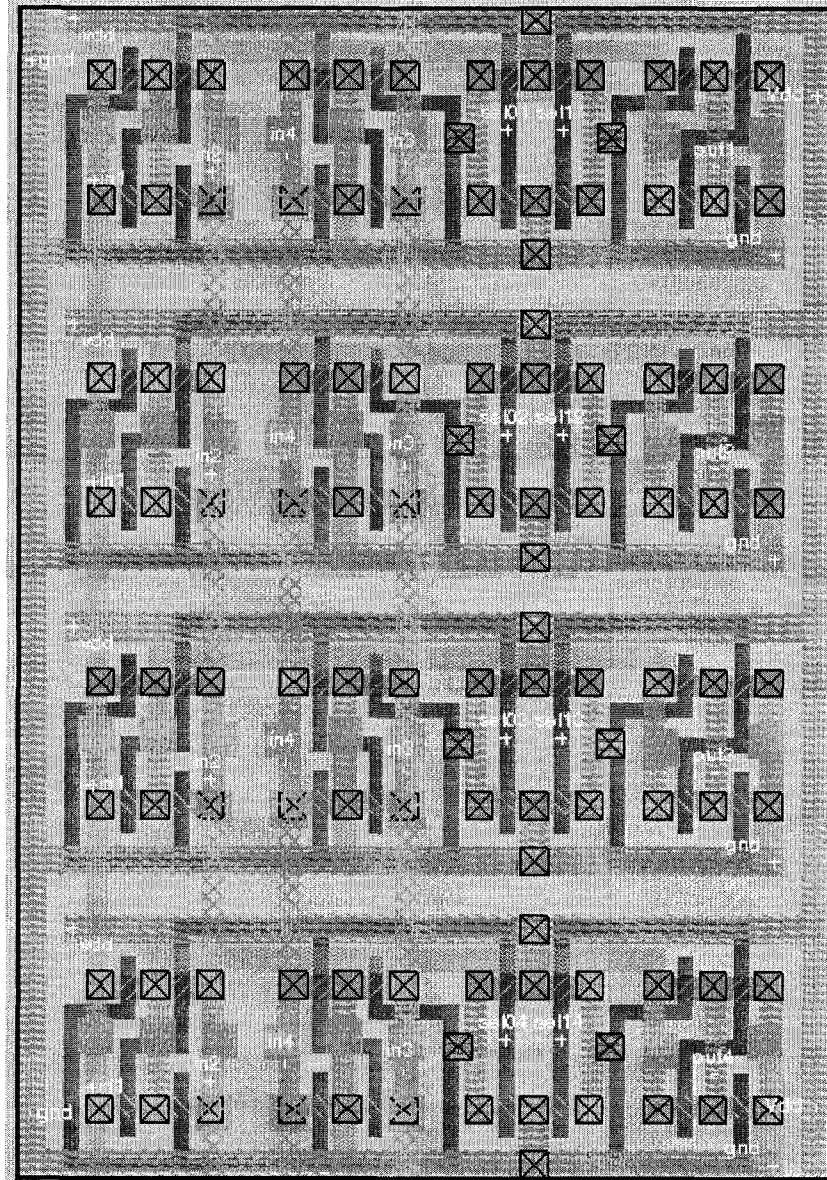


- [85] S.Q. Zheng, M. Yang, J. Blanton, P. Golla, D. Verchere, "A simple and fast parallel round-robin arbiter for high-speed switch control and scheduling," in *Proc. MWSCAS*, 2002, vol. 2, pp. 671-674.

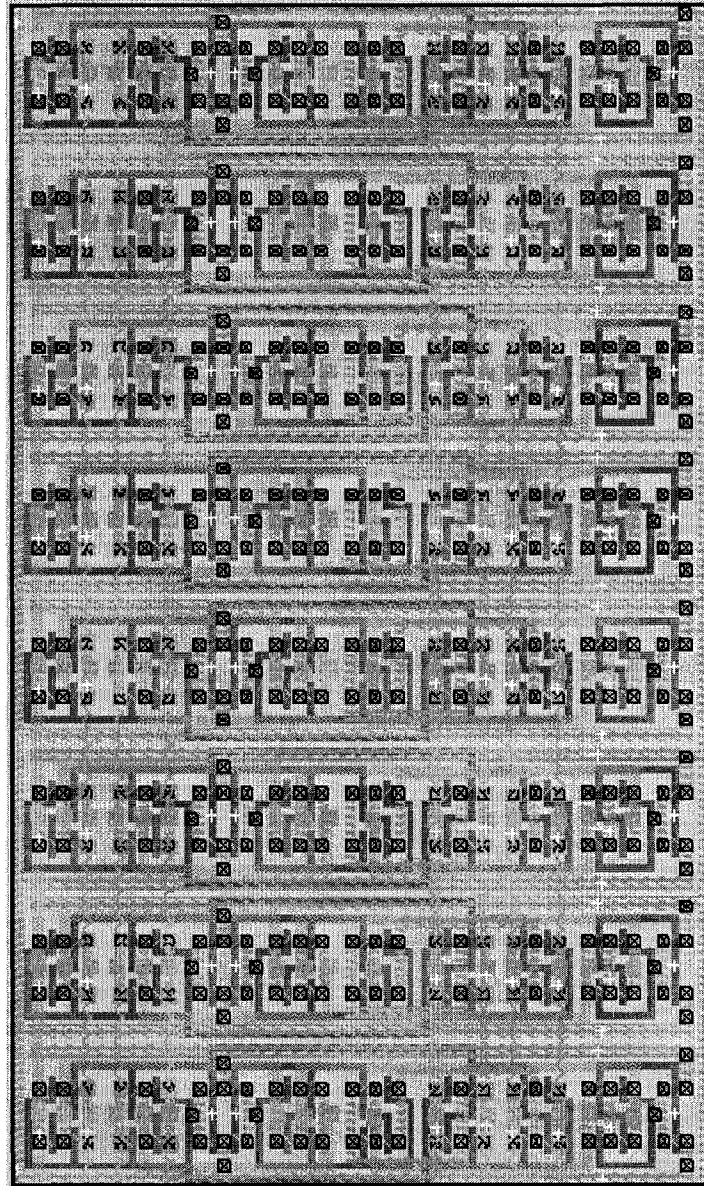
## APPENDIX



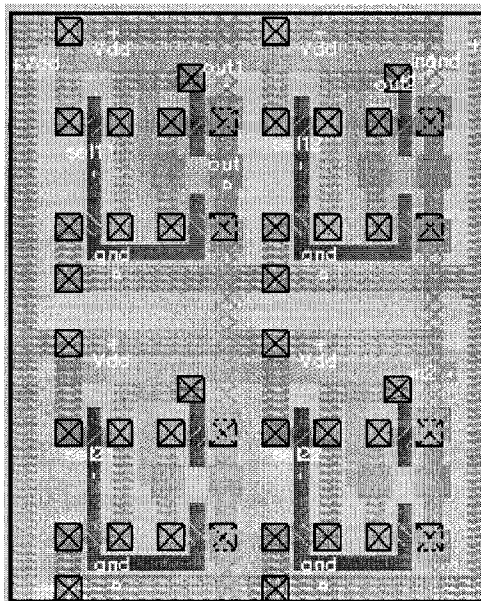
Layout of 2x2 Mux-based crossbar



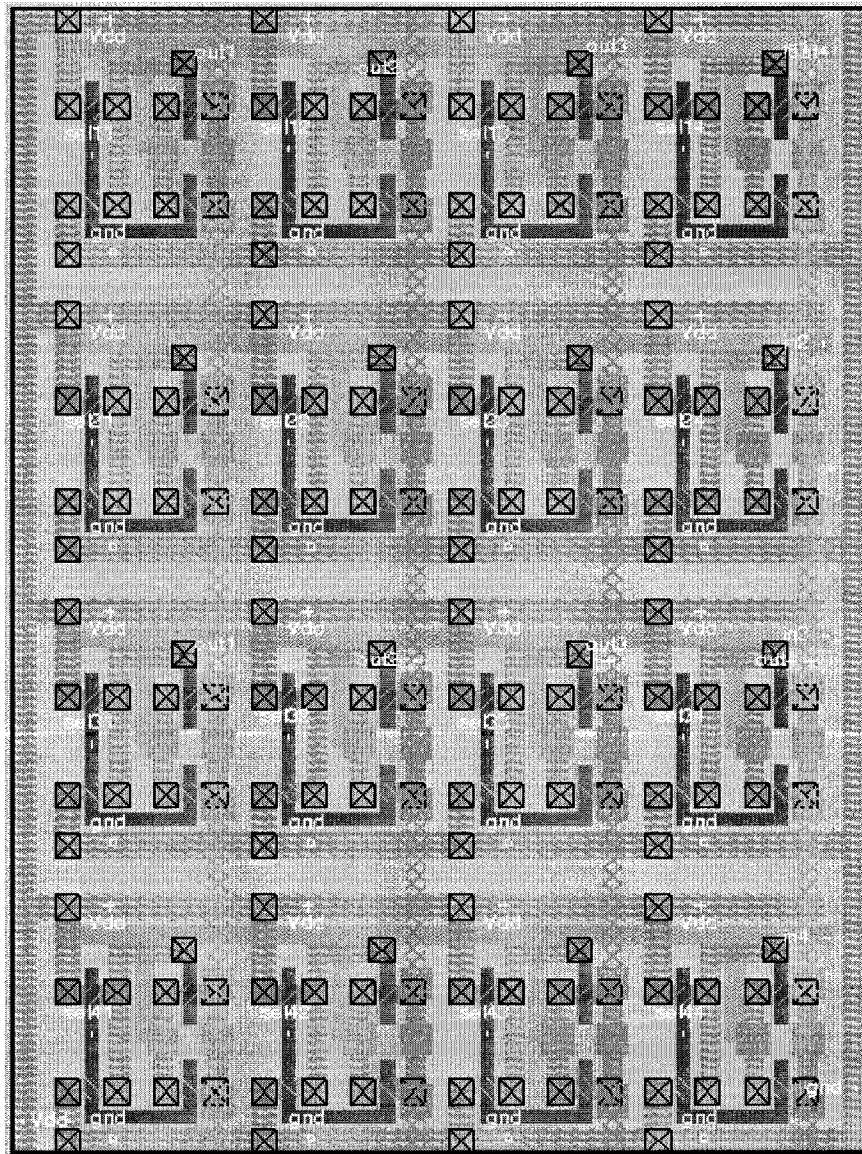
Layout of 4x4 Mux-based crossbar



Layout of 8x8 Mux-based crossbar

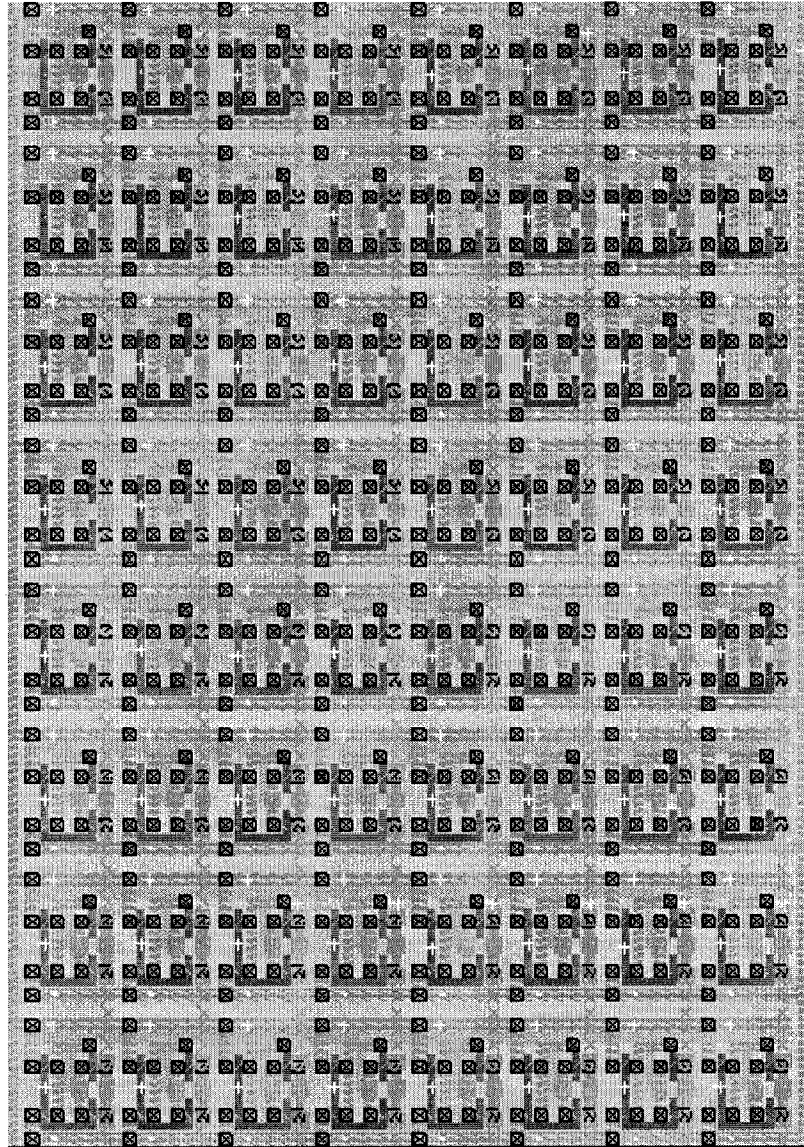


Layout of 2x2 Switch-based crossbar



Layout of 4x4 Switch-based crossbar





Layout of 8x8 Switch-based crossbar

## VITA

Graduate College  
University of Nevada, Las Vegas

Shankar Narayanan Neelakrishnan

Home Address:  
4247, Fairfax Circle, #1  
Las Vegas, NV 89119

Degree:  
Bachelor of Engineering, Electrical and Electronics Engineering, 2004  
University of Madras, India

Thesis Title: Design and Implementation of NoC Routers and their Application to PRDT-Based NoC's.

Thesis Examination Committee:  
Chairperson, Dr. Mei Yang, Ph.D.  
Committee Member, Dr. Venkatesan Muthukumar, Ph.D.  
Committee Member, Dr. Yingtao Jiang, Ph.D.  
Graduate Faculty Representative, Dr. Yoohwan Kim, Ph.D.