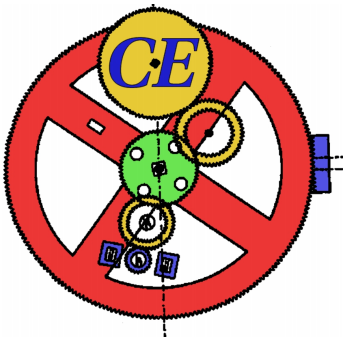


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2010

MSc THESIS

Hardware Design and Implementation of a Network-on-Chip Based High Performance Crossbar Switch Fabric

Turhan Karadeniz

CE-MS-2010-9

Abstract

High-performance routers have the task of transmitting traffic in between the nodes of the Internet, the network of networks that carries the vast amount of information among billions of users. The switch fabric is the key building block of every router, and various switch fabric architectures are used in the market products. The crossbar-based switch fabric architectures (both buffered and unbuffered) offer very high performances and are widely used for high-performance routers. However their cost grows quadratically with the input/output port count, since they require internal crosspoints (and buffers) for every input/output port pair.

Recently, a functional-level design of two novel Network-on-Chip based switch fabric architectures was proposed, Unidirectional NoC (UDN) and Multidirectional NoC (MDN), as a replacement of the buffered crossbar switch fabric architecture. In this thesis, we propose the hardware design and implementation of the aforementioned architectures for the FPGA platform. We further improve the routing and scheduling algorithms of these architectures for feasible hardware design. The synthesis and simulations are carried out over a wide range of switch sizes and traffic scenarios. The simulation results are also validated on the FPGA platform, by generating pseudo-random destination addresses for the packets on LFSR based test modules. The results show that UDN outperforms MDN in terms of throughput, whereas MDN offers greater performance-cost ratio. Both architectures offer scalability, flexibility and high performance, confirming the ideas in the original proposal.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Hardware Design and Implementation of a Network-on-Chip Based High Performance Crossbar Switch Fabric

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Turhan Karadeniz
born in Kastamonu, Turkey

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Hardware Design and Implementation of a Network-on-Chip Based High Performance Crossbar Switch Fabric

by Turhan Karadeniz

Abstract

High-performance routers have the task of transmitting traffic in between the nodes of the Internet, the network of networks that carries the vast amount of information among billions of users. The switch fabric is the key building block of every router, and various switch fabric architectures are used in the market products. The crossbar-based switch fabric architectures (both buffered and unbuffered) offer very high performances and are widely used for high-performance routers. However their cost grows quadratically with the input/output port count, since they require internal crosspoints (and buffers) for every input/output port pair.

Recently, a functional-level design of two novel Network-on-Chip based switch fabric architectures was proposed, Unidirectional NoC (UDN) and Multidirectional NoC (MDN), as a replacement of the buffered crossbar switch fabric architecture. In this thesis, we propose the hardware design and implementation of the aforementioned architectures for the FPGA platform. We further improve the routing and scheduling algorithms of these architectures for feasible hardware design. The synthesis and simulations are carried out over a wide range of switch sizes and traffic scenarios. The simulation results are also validated on the FPGA platform, by generating pseudo-random destination addresses for the packets on LFSR based test modules. The results show that UDN outperforms MDN in terms of throughput, whereas MDN offers greater performance-cost ratio. Both architectures offer scalability, flexibility and high performance, confirming the ideas in the original proposal.

Laboratory : Computer Engineering
Codenumber : CE-MS-2010-9

Committee Members :

Advisor:	Kees Goossens, ES, TU/e
Advisor:	Lotfi Mhamdi, CE, TU Delft
Chairperson:	Koen Bertels, CE, TU Delft
Member:	Rene van Leuken, CAS, TU Delft
Member:	Said Hamdioui, CE, TU Delft

Contents

List of Figures

List of Tables

Nomenclature

CICQ	Combined Input-Crosspoint Queueing
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HOL	Head-of-Line Blocking
IC	Integrated Circuit
INI	Input Network Interface
LAN	Local Area Network
LUT	Lookup Table
MDN	Multidirectional Network-on-Chip
NI	Network Interface
NoC	Network-on-Chip
ONI	Output Network Interface
PFU	Packet Forwarding Unit
RAM	Random Access Memory
RR	Round Robin
RRR	Round Robin Register
UDN	Unidirectional Network-on-Chip
VC	Virtual Channel

Acknowledgements

First of all, I would like to thank my supervisors, Kees Goossens and Lotfi Mhamdi, for their guidance all through the MSc project.

It was very helpful of Vlad-Mihai Sima, Yi Lu, and Dimitris Theodoropoulos to help me with the problems I have experienced with the design tools.

I would also like to thank Iria Varela Senin, for her work "Design of a High-Performance Buffered Crossbar Switch Fabric Using Network on Chip", which formed the basis for my thesis.

Last, but not least, I would like to thank Koen Bertels, Said Hamdioui and Rene Van Leuken for taking part in my graduation committee.

Turhan Karadeniz
Delft, The Netherlands
June 25, 2010

Dedication

I dedicate this thesis to

my mother, who showed me the simple ways to find joy in life,
my father, who got me interested in wondering about the ways of life,
my grandmother Tulay, who grew me up, in my beloved city of Istanbul,
my grandfather Rafet, who taught me the real meaning of affection and sentimentality,
my drop of life Damla, with whom I will make the best of life, as well as the best wine,

and my lifelong colleagues,
Onur Can Ulusel,
Stavros Tzilis,

and my lifelong mentors,
Nurdan Ugural,
Nino Carella,
Lou Ungemach,
Cengiz Agalar,
Erkay Savas,
Ilker Hamzaoglu,
Georgi Gaydadjiev,
and last, but not least, Serkan Sahin.

Turhan Karadeniz
Delft, The Netherlands
June 25, 2010

1 Introduction

This chapter describes the motivation and objectives of this thesis. First, it gives an overview of the background information that forms the basis for this research. Then, it describes the problem statement and the thesis contributions. Finally, it presents the outline for the following chapters.

1.1 Thesis Overview

The Internet has become the backbone of communication among the billions of users all over the world, connecting networks of different sizes, purposes and scopes. The Internet is defined as the network of networks (Figure 1.1), and it carries information and services among the vast number of nodes. The communication among these networks and nodes is realized by a broad and diverse body of electronic and optical technology. The routers have the task of transmitting the traffic to the destination nodes, through the best possible route. The links in between the routers and the nodes has to transmit the data as fast as possible, while the routers provide the bandwidth to ensure that they are not a bottleneck in the network communication.

The Internet is based on **packet mode communication**. All the information to be communicated is divided into suitably sized blocks, called **packets**. The routers connect two or more networks, decide the **routing path** of a packet that has been injected in the router and finally transmit the packet to the destination address (**packet forwarding**). These two essential tasks of the router are implemented by four basic modules, which are the **line cards, packet processing unit, routing unit,** and **the switch fabric** (Figure 1.2). The switch fabric, moreover, requires the implementation of a **scheduling unit**, which regulates and grants permission for the pairing of input/output ports.

The main design challenges for implementing switch fabrics include bandwidth, latency, scheduling algorithms, interfacing, and routing algorithms. Several switch fabric architectures have been proposed, including the **crossbar** [4], **shared-bus** [5] and **shared-memory** [6] switches, which deal with these design challenges in various ways (Figure 1.3). Crossbar switch is the dominant architecture in today's high-performance switches, due to a number of reasons. Crossbar switch is more scalable than the shared-bus and shared-memory; this is due to the limitations in bus transfer bandwidth and memory access bandwidth, respectively. Crossbar switch provides **point-to-point connections** and **non-blocking** properties, as well as supporting **multiple simultaneous transactions**, increasing the bandwidth and speed of the router.

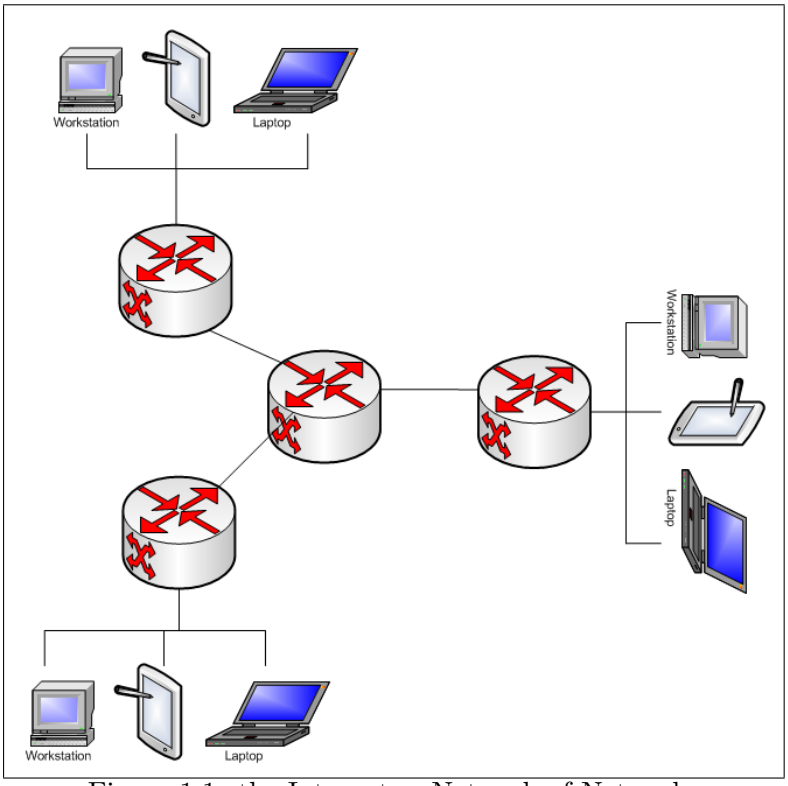


Figure 1.1: the Internet, a Network of Networks

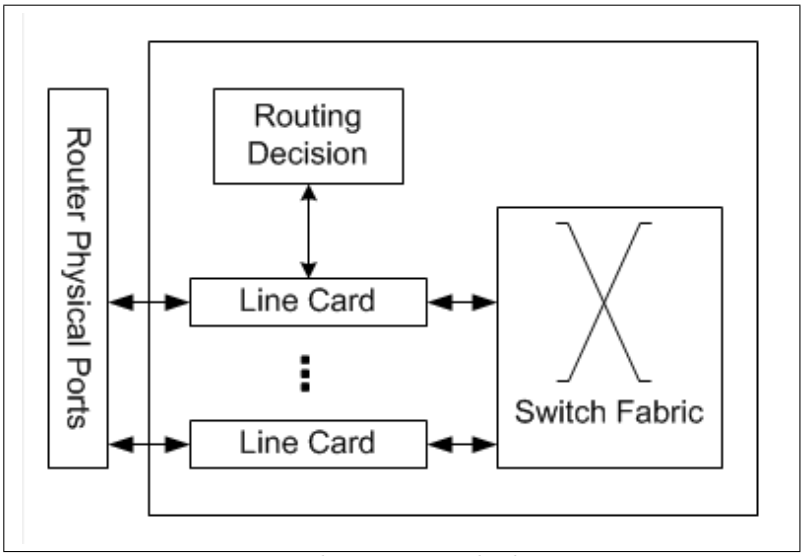


Figure 1.2: the Router Block Diagram

The crossbar switches are further divided into **unbuffered** [7] and **buffered** or **combined input-crosspoint queueing (CICQ)** [8][9] crossbar switches. The unbuffered crossbar switches have the advantage of low cost, because of using no internal buffers;

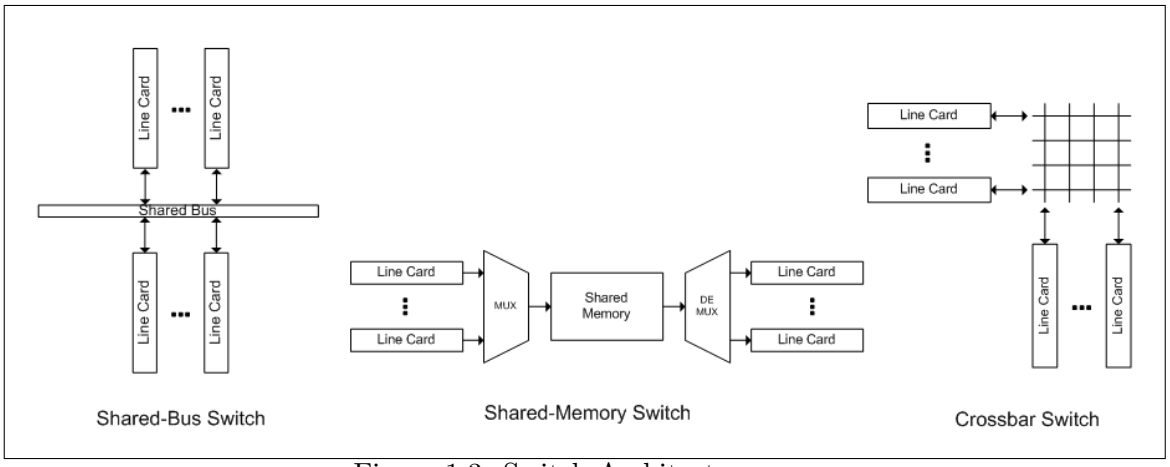


Figure 1.3: Switch Architectures

however the scheduling unit gets very complex and needs to be centralized. On the other hand, the buffered crossbar switch architectures require **dedicated internal buffers**, which are instantiated per all input/output combinations, resulting in the area cost to grow quadratically with the port count; however the scheduling unit is distributed and simpler.

1.2 Motivation

1.2.1 Problem Statement

The proposal in [1], [10], and [11] studies the performance of Network-on-Chip based crossbar switch fabric, targeting greater scalability and flexibility, as well as greater performance per hardware cost, compared to buffered crossbar switch fabric (Figure 1.4). [1] proposes functional-level design of two NoC based switch fabric, Unidirectional NoC (UDN) and Multidirectional NoC (MDN).

In this thesis, we address to the problem of hardware design and implementation of the NoC based switch fabric architectures, and try to validate and prove the claims of the original proposal in hardware. The hardware implementation mandates the altering of some design parameters, constraints and algorithms, during the transition from functional-level to register-transfer level design. In particular, we wish to investigate the performance and cost of the NoC based switch fabrics over a wide range of switch sizes, to explore the hardware implementation feasibility.

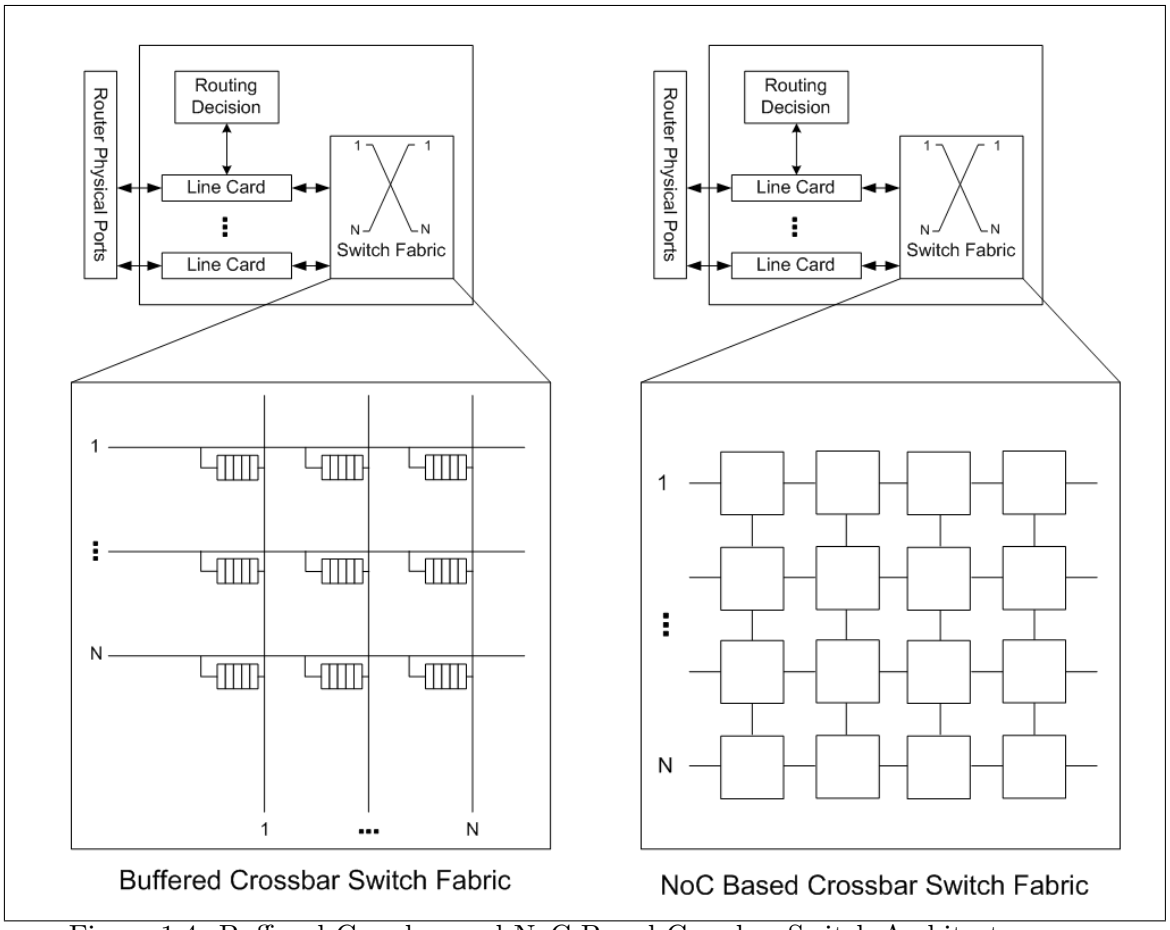


Figure 1.4: Buffered Crossbar and NoC Based Crossbar Switch Architectures

1.2.2 The Thesis Contributions

In this thesis, we carry out the hardware design and implementation of NoC based crossbar switch fabric proposed in [1], and investigate the performance and cost through synthesis and simulations over a wide range of switch sizes. The hardware design is implemented in RTL, using VHDL. The simulation results are also validated on the FPGA platform, by generating pseudo-random destination addresses for the packets on LFSR based test modules.

In the thesis, we have explored the scalability of the UDN and MDN switches, over uniform and non-uniform (unbalanced) traffic scenarios. We have altered the MDN routing algorithm in the original proposal, for better performance and simpler implementation. We have further proposed a method to improve load balancing by randomly assigning the T-Value parameter of the routing algorithm of UDN and MDN. Finally, we have proposed various arbitration schemes based on Round Robin scheduling, and explored their performances.

1.3 Thesis Outline

The remainder of this thesis is organized as follows:

In Chapter 2, we present the background information and the related literature. This chapter includes the detailed explanation of the terminology used throughout the thesis, various architectures that historically have been the milestones in the network switch design, Network-on-Chip related information, and discussions why a certain design decision has been preferred over the others.

In Chapter 3, we present the hardware design of the Unidirectional NoC (UDN) and Multidirectional NoC (MDN), including the Input Buffers, Network Interface and Routers. The routing and scheduling algorithms and their hardware implementations are discussed in Chapter 4.

In Chapter 5, the synthesis results are presented for the individual modules that constitute the UDN and MDN switches, as well as different sized UDN and MDN switches. The synthesis results include the operational frequency and the 2-tuple resource cost (Number of Slice LUTs, Number of Slice Registers) of the modules.

Simulation results of UDN and MDN switches are presented in Chapter 6. The switches are tested under two different traffic types: the Bernoulli Uniform and Unbalanced (Weighted) Traffic. In addition, our proposals for improving the scheduling and load balancing are simulated in order to explore their positive and negative aspects.

In Chapter 7, we define the performance and cost functions for the UDN and MDN architectures, and present the performance / cost analysis for both architectures, over a wide range of sizes and various traffic loads. We conclude this chapter with a comparison of the architectures.

In Chapter 8, we present the in-circuit verification, which is an important step of reconfigurable hardware design; it involves mapping the final design on the reconfigurable platform (FPGA) and carrying out the verification/validation of the results and the timing analysis.

Finally, we conclude the thesis in Chapter 9, giving a detailed observation on the results we have obtained and point out the possible future work.

Background Information & Related Work

2

This chapter includes the detailed explanation of the terminology used all through the thesis, various architectures that historically has been the milestones in the network switch design, the emergence of network-on-chip evolving from computer networks as a new paradigm for on-chip communication, and some communicational protocols.

2.1 Introduction

The Internet is a network of networks, carrying vast quantity of information and services among the billions of users all over the world. Internet services such as WWW, email, VOIP, IPTV, e-commerce, e-banking, FTP etc. are increasingly used on daily basis [12], and therefore require increasing **network bandwidth**, such that the great quantities of data can be communicated in between the servers and users. **High-performance routers** have the task of transmitting traffic in between the nodes of the Internet. The routers need to support high bandwidths, in order not to become the bottleneck in the network communication.

The Internet is based on **packet mode communication**. All the information to be communicated is divided into suitably sized blocks, called **packets**. The routers connect two or more networks, and perform two important functions: **routing path determination**, computing the route of a packet that has been injected in the router, and **packet forwarding**, transmitting the packet to the destination address. **The line cards, routing unit,** and **the switch fabric** are the key building blocks of a router, which implement the two essential functions of the router. When a packet is injected into the router, it is first accepted by the inbound network interface (input line card). Once the routing path is computed on the routing unit, it is forwarded via the switch fabric to the outbound interface (output line card) and finally ejected from the router, to its final destination [13].

The design of the switch fabric, the module responsible of packet forwarding, constitutes an important part of the network router design, and therefore it remains to be an open research problem. Various switch fabric architectures are used in the market products, like Vitesse's GigaStream and IBM's Power Packet Routing Switch [14].

2.2 Switch Fabric Architectures

The development of switch fabric architectures has shown a trend from the **shared resources** towards **dedicated resources**. Some of the reasons for this trend include:

- The need for higher bandwidths. We live in a web-dependent world, where the use of Internet services, such as email, VoIP, streaming audio and video, search engines and online encyclopedias, become a must even in our daily lives. All these services require high data rates, and therefore higher bandwidth in the routers become crucial for data transfers.
- Developments in the integrated circuits (IC) technology, which have led to the point where very complex systems could be implemented on a single chip, resulting in low cost routing solutions that incorporate hardware-software co-design to yield high performance routers [13].

The important parameters in designing a switch fabric are **throughput, packet delays, amount of buffering, scalability, complexity of the implementation** and **packet loss**. The shared-bus, shared-memory, crossbar architectures are the basic approaches that shape the switch fabric design; the proposed solutions are either a variation or combination of these architectures. As a result, it is important to describe the characteristics of these approaches [13].

The shared-bus switch fabric (Figure 2.1) is the simplest among the router architectures. It is implemented by a shared single medium over which all data is transmitted. Statistical multiplexing is used to allocate the bus for an input/output port pairing, since the Internet is based on packet mode communication. **Statistical multiplexing** might either serve the packets in the first-come-first-serve fashion, or might deliver the packets according to a scheduling discipline (such as **Round Robin**) for fair queuing. The bus is the bottleneck of the communication, since all the traffic flows over it. It is reliable and simple to implement. As the number of input/output ports increase, scalability becomes an important issue, as it gets more difficult for the bus to support the total bandwidth.

The shared-memory switch fabric's typical architecture is shown in Figure 2.2. The shared-memory architecture is implemented by a dual-port, shared and centralized memory which is used as an output buffer. The inbound packets are multiplexed into the centralized buffer, and the outbound packets are demultiplexed into the output line cards. The advantage of the approach is that the buffer can be statistically shared, according to the buffering needs. The disadvantage is that the memory access speed becomes the bottleneck, resulting in scalability and bandwidth issues.

The third approach, crossbar switch fabric (Figure 2.3), relies on dedicated resources, as opposed to the shared resources of the first two architectures. In a crossbar, there exists a crosspoint for any input/output port combination. The crossbar switches are very good for their scalability and non-blocking properties. Since there is a crosspoint for

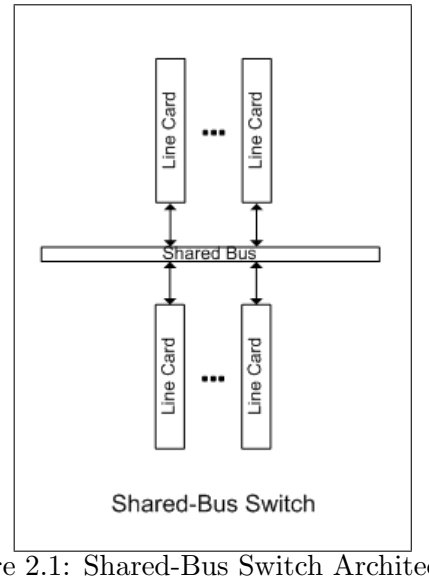


Figure 2.1: Shared-Bus Switch Architecture

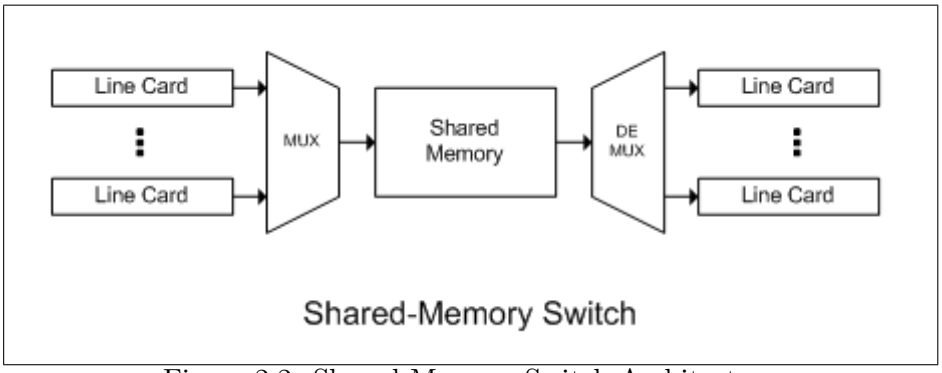


Figure 2.2: Shared-Memory Switch Architecture

every input/output port pair, it also supports multiple simultaneous transactions. The crossbar switches are further divided into **unbuffered**, and **fully-buffered switches**.

Some variations and combinations of the above approaches, such as partially-buffered crossbar switch fabric [15] and Bufferless Clos-Network Switches [16], are also available. [15] proposes a partially-buffered architecture that is comparable to the cost of unbuffered switch, and to the performance of fully-buffered switch. [16] proposes a Memory-Space-Memory architecture which is a combination of shared-memory and bufferless crossbar architectures.

Recently, a functional-level design of two novel Network-on-Chip based switch fabric architectures were proposed, Unidirectional NoC (UDN) and Multidirectional NoC (MDN), as a replacement of the buffered crossbar switch fabric architecture proposed in [1]. With this novel architecture, the trend goes back towards shared resources, in exchange of scalability.

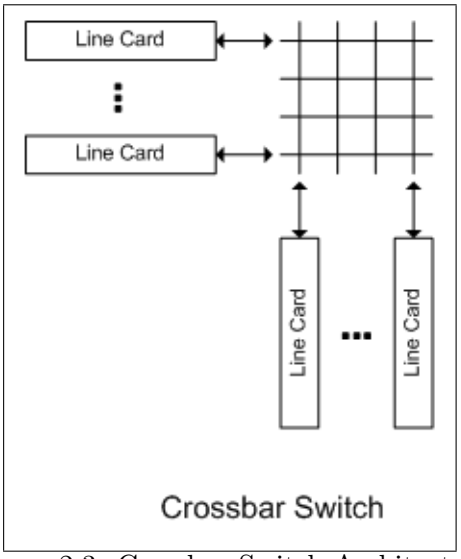


Figure 2.3: Crossbar Switch Architecture

In the next section we explain the emergence of Network-on-Chip as an on-chip communication paradigm.

2.3 Emergence of Network on Chip (NoC)

On-chip communication has evolved, in accordance to the increase in number of elements that are present in a certain system-on-chip. As the number of modules increased, the difficulty of implementing the communication has increased as well. The scalability of the communication circuitry became an important part in the design effort. With these aspects, the evolution of on-chip communication resembles of the switch fabric; the main difference being that the communication on-chip is in **circuit mode**, as opposed to **packet mode**. This means that a dedicated circuit (or channel) has to be established in between the nodes, as if they were physically connected via an electrical circuit [17].

The simplest methodology, involves the communication of two IPs, via a dedicated **point-to-point connection**. This communication involves a hand-shake protocol; a valid signal indicating that the initiator has set the data for the target, and an accept signal, indicating that the target has used the data. In this methodology, the modules need to have a connection to each module they need to communicate with, exponentially increasing the implementation cost. The scalability and re-use of these types of communicational circuits are very low.

A remedy to the high costs and re-use issues of point-to-point communication is to communicate over a shared single communication channel, a **bus** [18], which all modules are connected to. In this case only one of the modules can access this channel at a given time, resulting in a system bottleneck. The modules that are in communication

at a given time seem as if they are hard-wired to the bus that they are communicating through, while the rest of the modules are not. To realize this, there is the need of an arbitration (scheduling) system, to grant the communication rights to the nodes over the bus. On the other hand, the bus communication lacks scalability, like the point-to-point communication.

Verifying the communication circuit repeatedly even for minor changes in the system quickly become a major burden in the design effort, and therefore designing for scalability becomes crucial [19]. **Network-on-Chip (NoC)** is a solution to the scalability issue in on-chip communication. NoC communication has a different structure from the other on-chip communication architectures, because it is based on packet mode communication, like a network router. The communication in NoC is realized on a router matrix, where the nodes are connected to each other with point-to-point connections [20]. This kind of a structure makes it possible for new routers to be added in the matrix very easily.

In on-chip communication, point-to-point is one end, representing singularity in connections, whereas NoC is the current other end, emphasizing scalability and abstraction among many others. **Abstraction** is another important aspect of NoC [21]; because the router communication is based on packets, the computational part is completely separated from the communicational part. Abstraction implies **composability**, which means changing only some parts of NoC without changing and re-verifying rest of the system. Abstraction layers are implemented to provide services for the higher abstraction levels to use, hiding the details of implementation, yielding composability [21].

2.4 Network on Chip

Network on chips, a relatively new concept in system-on-chip communication, borrows many ideas from the computer networks, the domain in which the research on routers and packet switching has matured. However, these ideas need to be adapted in NoC as well, since there is no direct translation of these methodologies [22].

Figure 2.4 shows a network-on-chip, in the form of an N-by-N router matrix. Computational cores are connected to **network interfaces**, which are in turn connected to the routers. Different network topologies such as **mesh, torus, express cubes**, etc. can be used to enhance the connections in between the routers, such that the **diameter of the network** (largest minimal hop count) can be decreased [23]. Some other important concepts in NoC are **routing, flow control, buffer management, quality of service** and **network interfaces**.

The **router** is the most important building block of a network-on-chip. **Routing decision** is about finding the path in between a start and ending point in NoC, which can be in a **deterministic, oblivious** or **adaptive** fashion [17]. In deterministic routing, the path in between the source and destination are always the same. In oblivious routing, any of the possible routes is chosen, regardless of the network state. In adaptive routing, the network state is taken into consideration while computing the **routing path**. The

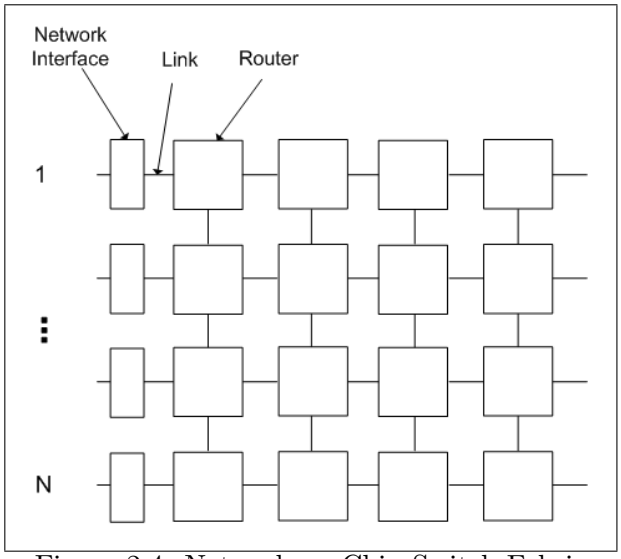


Figure 2.4: Network-on-Chip Switch Fabric

routing path can be computed **all-at-once** in the NI, or **incrementally** (step by step) on the routers. As the packets are transmitted over the routers, **contention** (more than one packets competing over a certain route) and **congestion** (decrease in throughput due to contention) become the main routing problems that need to be avoided. As a solution, **flow control** needs to be applied in either a **bufferless** or **buffered** fashion. Bufferless flow control deals with the routing problems by either dropping both of the packets; dropping one of the packets; or misrouting one of the packets. Buffered flow control involves use of large buffers in the routers to store one of the packets, which increases the size and cost of the circuit, as well as adding **buffer management** overhead [17], in return of **lossless** routing.

The packets' data structure is organized in two sections, the **header** and the **payload**. The header carries the routing related information and instructions, whereas the payload carries the actual data packet is delivering to the destination. Large network packets are broken into smaller pieces called **flow control digits (flits)**, for serializing the packet transmission in between the routers.

In buffered flow control scheme, the **switching mode** determines when a packet is forwarded in between two routers. In **Store and Forward** mode [24], a packet is forwarded when the current router has received the whole packet, and the next router has the space for the whole packet. In **Virtual Cut Through** mode [25], a packet is forwarded when the current router has received the flit, and the next router has the space for the whole packet. In **Worm Hole** mode [26], a packet is forwarded when the current router has received the flit, and the next router has the space for the flit.

Buffered flow control solves the contention by forwarding one of the packets and storing the other, which requires the routers to have buffers in various schemes. **Input Buffering** is the simplest and cheapest scheme, with the worst performance among

others; it is implemented either by **first-in-first-out (FIFO)** buffers per input port or a shared-memory [27]. **Output Buffering** is an efficient scheme, however it is the most expensive since it is implemented by FIFO buffers per input/output port pair. **Virtual Output Buffering** [28] is logically the same as Output Buffering, however implementation-wise it uses dual port rams per input port, virtualizing the buffers per input/output port on a single ram. Input and (Virtual) Output Buffering schemes suffer from **head-of-line blocking (HOL)**, in case of contention. Some architectures consist of combinations of the above schemes [29].

If a packet is waiting for an event to occur, but the event will never occur due to some blocking in the switch, this is called a **deadlock**. In Figure 2.5, we exemplify this problem, on a mesh based topology, with buffered flow control. When HOL occurs in a circular loop, the packets cannot be forwarded to the next routers, because all of the input buffers are already full and are blocking each other. A proposal to avoid deadlocks suggests the use of virtual channels, implemented as parallel buffers, to yield deadlock-free network [30][31].

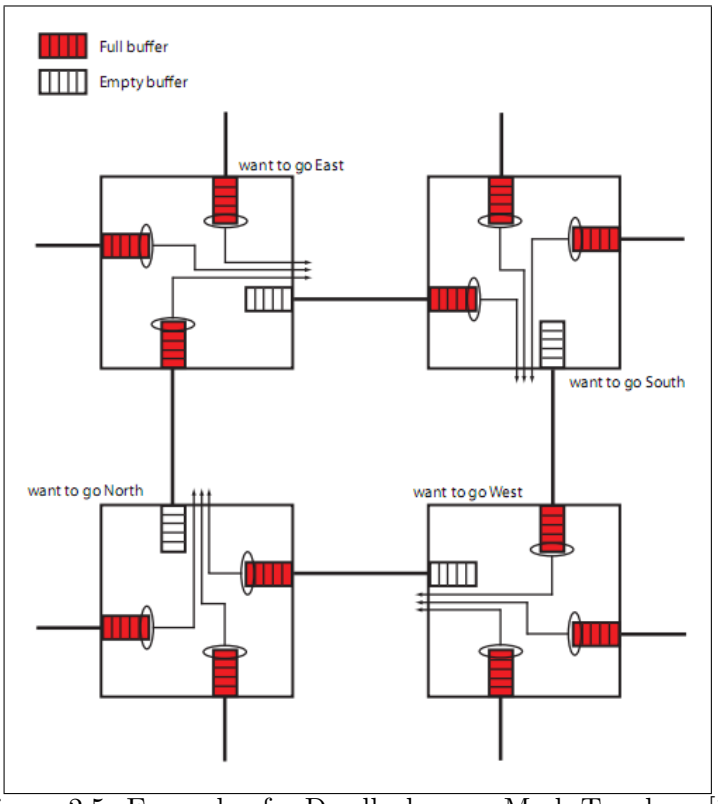


Figure 2.5: Example of a Deadlock, on a Mesh Topology [1]

The **network interface** acts as an **abstraction layer** between the computational unit's interface and NoC interface (Figure 2.6). Since on-chip communication is based on circuit mode, the data has to be packetized in the **packetization** unit of the network interface. Once the packets are generated, they are transmitted to the next router, in

equally sized flits.

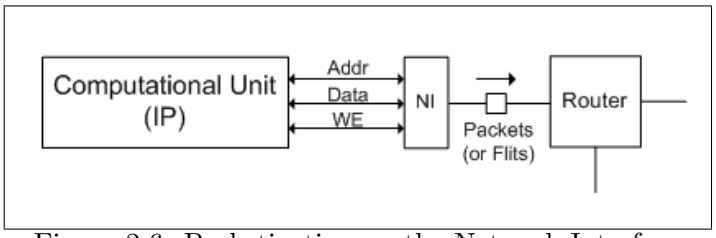


Figure 2.6: Packetization on the Network Interface

Once the packet (or flit) is received by the router and its routing path has been computed, the packet needs to be forwarded to the next router in the NoC switch. However, a NoC router has multiple input ports competing for an output port, resulting in contention. A **scheduling algorithm** computes which packet has to be forwarded prior to the other packets, and therefore which input port is to be connected to a certain output port. The **arbiter** makes a link in between the input and output port, according to the output of the scheduling algorithm, such that the packet is transmitted. In this thesis, we will refer to the scheduling and arbitration terms interchangeably. The arbitration on NoC is very similar to the arbitration in the bus communication. The scheduling algorithm might be implemented as a variation of **Random** [32], **Priority** [33], or **Round Robin** [34] algorithms among many others. Round Robin offers fair scheduling, assigning each resource equal usage in circular order, which results in starvation-free system.

As mentioned earlier, the buffered flow control requires **buffer management**: this is to inform the previous router about the availability of the buffers in the current router. This is also called **backpressure**, meaning that the current router informs the previous routers when they should stop transmitting packets (or flits). There are three types of flow control mechanisms that are commonly used, to provide backpressure: **credit-based, on/off** and **ack/nack** [35]. In the credit-based scheme, the sender keeps track of the empty buffer space in the receiver. It sends data only when remote space is greater than 0, and the credits are returned by the receiver when its buffer space becomes available. If the credit can only be equal to 1 or 0, then the scheme is called **valid/accept**. In on/off scheme, only the changes in the buffer are sent, like when the buffer is full and available. In the ack/nack scheme, the packets (or flits) are sent optimistically without keeping track of the buffer availability; if the buffer is not available, the packet is dropped, and nack signal is set to inform the sender that it has to be sent again [17].

The communication through the NoC is **pipelined** automatically due to the nature of the routing and point-to-point communication in between the routers. In this way, the critical path is restricted to the control signals in a router, improving the scalability of the switch. Once the routing decision has been made, and the first flit has been issued to the next router, all the other flits are going to be delivered each cycle. This fact increases the throughput of the system [20].

Some acclaimed implementations of NoC include **Aetherial** [36], **Nostrum** [37],

Xpipes [38], **Intel 80 Core NoC** [39], and **Mango** [40]. They all make different decisions in terms of the schemes explained in the previous paragraphs, to achieve their design goals.

NoC, a paradigm of on-chip communications, with its basic concepts borrowed from computer networks, is proposed to be applied back to its original domain, to remedy some shortcomings in the crossbar switch fabric design. In regard to this matter, the basic building blocks of NoC, including the buffers, flow control, arbitration and routing units need to be used in the correct combination of schemes, to be able to provide competitive solutions.

2.5 Communication Technologies, Protocols & Packets

ISO/OSI 7 layer model (Figure 2.7) is a reference model which divides a communication system into smaller parts called layers [41]. In this way, the layers are abstracted from each other. Each layer offers services to higher layers, using the services of the lower layers. All the communicational protocols are placed into one of the layers. The Internet also uses a set of protocols called TCP/IP, which is very similar to ISO/OSI 7 layer model with its four abstraction layers.

Layer 7	Application
Layer 6	Presentation (Data Representation)
Layer 5	Session (Interhost Communication)
Layer 4	Transport (Flow control)
Layer 3	Network (Routing, (De)packetization)
Layer 2	Data Link (Transport of frames/flits)
Layer 1	Physical (Transport of single bits)

Figure 2.7: ISO/OSI 7 Layer Model, the Abstraction Layers

Ethernet technology used in **local area networks (LANs)**, **internet protocol (IP)** technology used in packet-mode internetworking, and **asynchronous transfer mode (ATM)** technology used in simultaneous digital transfer of video, audio, data, etc. as in ISDN systems are among the most known examples of technologies used in communication, and their similarly named protocols are categorized under different layers of OSI and/or TCP/IP models. Ethernet and IP protocols use **variable-size packets**, whereas ATM packets (cells, in common usage) are **fixed size** [2]. ATM protocol involves the properties of both circuit mode and packet mode networking, and therefore it is suitable for wide area data networking as well as real-time media transport. Moreover, Ethernet and IP packets can be encapsulated in ATM packets [42][43].

In the light of the above information, we chose ATM packets as the input/output packet type of our switch. For the other packet types, we would only have to change the (de)packetizing unit in the network interface; therefore this is not a crucial design

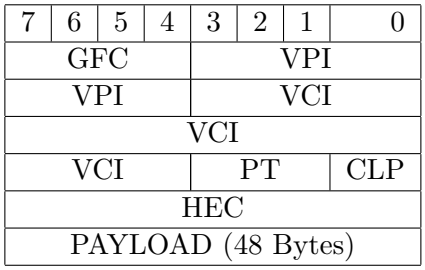


Figure 2.8: Diagram of the UNI ATM Packet, where the top row denotes the number of bits. [2]

decision. The ATM packet (Figure 2.8) is 53 bytes in length, where 5 bytes constitute the header section and 48 bytes constitute the payload.

3 UDN/MDN Hardware Design

Recently, functional-level designs of Unidirectional NoC (UDN) and Multidirectional NoC (MDN) have been proposed as a replacement of the buffered crossbar switch fabric [1]. In this chapter, we explain how these functional-level designs can be implemented in hardware for reconfigurable platforms, specifically Field-Programmable Gate Arrays (FPGAs). The routing and scheduling algorithms and their respective implementations are presented in Chapter 4.

3.1 UDN/MDN Overview

Block diagrams for UDN and MDN switch architectures are presented in Figure 3.1 and Figure 3.2. The main difference among the two switches is how their input/output pins are placed in the layout. In UDN, the input pins are placed on the west side of the layout, whereas the output pins are on the east side. On the other hand, in the MDN switch, the pins are placed all around the peripheral, where input and output pins are next to each other. In Figure 3.1, it can be observed that the northmost row routers of UDN Switch have two input (West, South) and two output (East, South) ports; similarly, the southmost row routers have two input (West, North) and two output (East, North) ports. The central UDN Routers have three input (West, South, North) and three output (East, South, North) ports.

UDN and MDN switch architectures have the same NoC specifications. The switching mode is in **store and forward** scheme, and **input buffers** are used for the buffered flow control, implemented by **FIFOs**. The scheduling is based on variations of the **round robin** scheme. In the original proposal [1], the buffer management is credit-based, however in our design we prefer the **valid/accept** scheme instead; the drawback of credit-based buffer management is that for every flit transfer, a credit is sent to the previous router, resulting in a communication overhead [35].

The size of the UDN switch, as shown in Figure 3.1, is defined by the 2-tuple (N, M), where N denotes the number of input/output ports, and M denotes the number of router columns. Some restrictions apply to (N, M) values: $N \in \mathbb{N}$, and $N \geq 2$; $M \leq N$ and $M = 2^m$, where $m \in \mathbb{N}_0$. The restrictions on M are caused by the routing algorithm involving Modulo M operations (discussed in Chapter 4). Because Modulo M operation requires division in case $M \neq 2^m$, but it is a simple bit-selection operation in case $M = 2^m$, we can avoid the extra cycles caused by the division operation by applying this restriction. With some minor modification in the scheduling algorithm, $M = N - 1 \Rightarrow N$

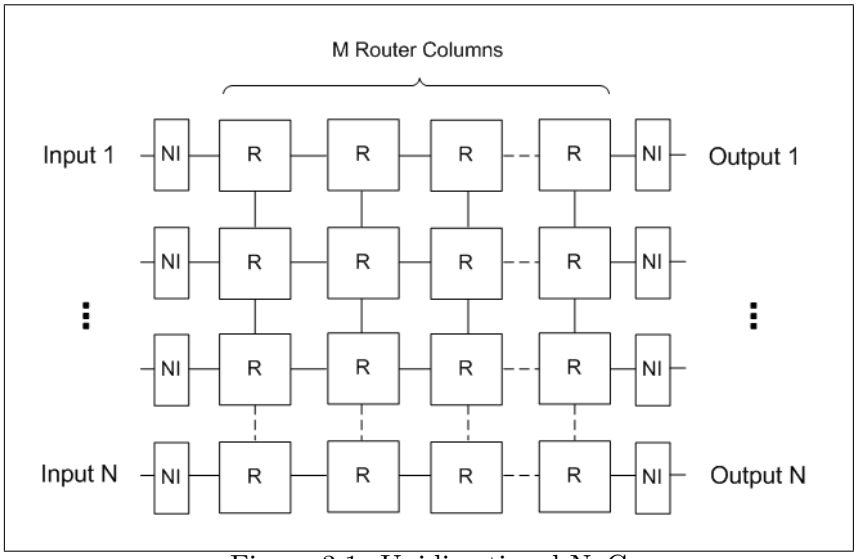


Figure 3.1: Unidirectional NoC

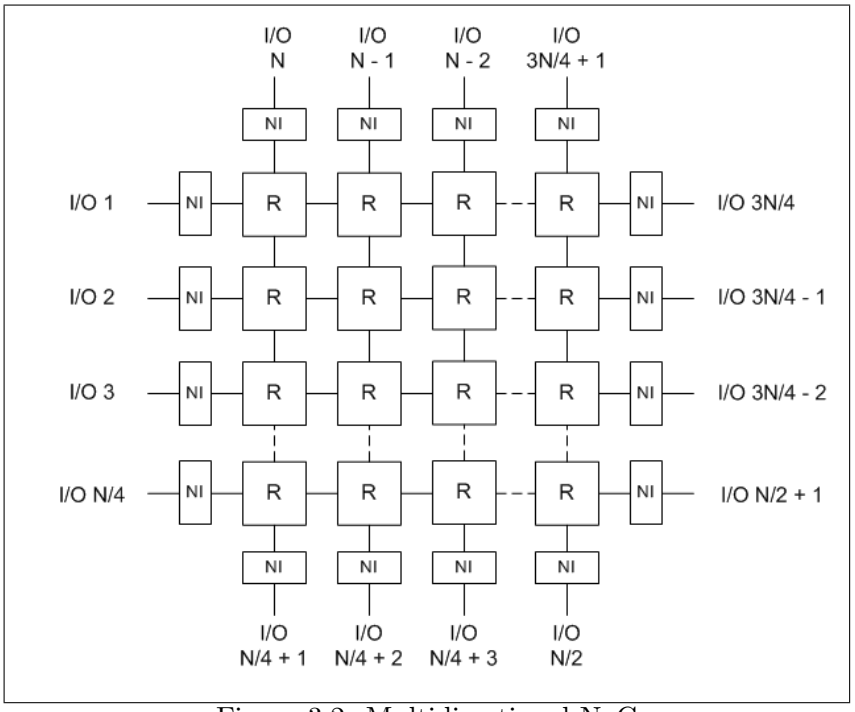


Figure 3.2: Multidirectional NoC

$= 2^n$, where $n \in \mathbb{N}_1$ is also a possible (N, M) combination; this is further explained in Chapter 4, in the UDN routing algorithm. The number of routers in the UDN switch is equal to $N \times M$.

The size of the MDN switch, as shown in Figure 3.2, is defined by N, which denotes

the number of input/output ports. Because the input/output ports are placed around the peripheral of the switch, some restrictions apply to N: $N = 4 \times n$, where $n \in \mathbb{N}_1$. The input/output ports are placed counterclockwise, starting from the West side of the layout. The ports in between 1(N/4) are placed on the West, (N/4 + 1)(N/2) on the South, (N/2 + 1)(3N/4) on the East and (3N/4 + 1)N on the North side of the layout. The number of routers is equal to $(N/4)^2$.

In the UDN and MDN switch design, the same **network interface, flow control unit,** and **router input buffer (FIFO)** modules are used. The UDN and MDN routers are different, due to the different number of ports; the top-level switch designs are also different due to the placement of the switch pins. In the next sections, we explain our design decisions for the UDN and MDN building blocks and top-level switches.

3.2 U(M)DN Packet Organization

U(M)DN Packet is a wrapper for ATM packets, such that they can be routed in the UDN and MDN switch fabrics. The routing algorithms for UDN and MDN, as proposed in [1], require different input parameters (discussed in Chapter 4); therefore it is not possible to use the same packet type in both of the switches. In Chapter 4, we propose a new MDN routing algorithm, which makes it possible to use the same packet type in UDN and MDN switches, with no abundant data fields. The U(M)DN packet consists of the header and payload sections (Figure 3.3). The payload is where the ATM packet is stored.

The header is composed of the routing information, including **Valid Bit, Unicast/Multicast Bit, T-Value,** and **Destination Address(es) Bitmap**. **Valid Bit** is set when the ATM packet is wrapped into the U(M)DN packet (packetized) in the network interface and is used to check if FIFO pointer is pointing to a valid packet; this bit is only an extra precaution, because our FIFO design have mechanisms to control if the buffer is empty or has valid packets to be forwarded. **Unicast/Multicast Bit** determines if the packet has a single (**unicast**) or multiple (**multicast**) destination addresses. **Destination Address(es) Bitmap (DAB)** is the field where the destination address(es) is(are) stored; in the unicast traffic, there is a single address, which is encoded in unsigned binary form, whereas in the multicast traffic, the destination addresses are bitmapped onto this field. Please note that the U(M)DN packet header includes the latter two data fields, in order to be compatible with the proposal in [1] that supports multicast traffic, even though our UDN/MDN switch implementations do not involve multicast traffic, which is left as future work. **T-Value** is used for load balancing, in the routing algorithms of both of the architectures, and it will be further explained in Chapter 4.

The width of T-Value and DAB is determined according to the switch size. In UDN, T-Value requires a width of LOG2(M) bits, and DAB, a width of N bits. In MDN, T-Value requires a width of LOG2(N/4) bits, and DAB, a width of N bits. Considering an UDN switch (N, M) = (64, 64), then T-Value is 6 bits, and DAB is 64 bits. Considering

7	6	5	4	3	2	1	0
T-Value						U	V
Destination Address(es) Bitmap							
PAYLOAD (ATM CELL) (53 bytes)							

Figure 3.3: U(M)DN Packet Structure, where the top row denotes the number of bits.

an MDN switch (N) = (64), then T-Value is 4 bits, and DAB is 64 bits. The resulting U(M)DN Packet structure is shown in Figure 3.3.

In order to have the same size of packets in the simulations, we have fixed the U(M)DN packet size to 63 bytes, where 53 bytes are the payload, and the remaining 10 constitute the header. We divide the U(M)DN packet into seven 9-byte flits, and serialize the transfer between the routers.

3.3 Input Buffers (FIFOs)

Before diving into the hardware design and implementation details of NI and Routers, it is necessary to explain the general structure of the input buffering hardware, since the same architecture is used with small variations in NI and Routers. The input buffer we have chosen to implement is based on the **register-based synchronous FIFO buffer**, in the form of a **circular queue** [3]. Instead of using the embedded block RAMs on the FPGA, we used the flip-flops (slice registers), for better cost analysis, as discussed in Chapter 5.

The circular queue is implemented as presented in Figure 3.4. The initial register position is marked by the **Read Pointer (RP)** and **Write Pointer (WP)**. During a write operation, the data is written into this register, and the WP is incremented. In the same way, after a successful read operation, the RP is incremented to point to the following data. **Status register (SR)** is incremented after each write operation and decremented after each read operation. If SR = 0, then the buffer is **empty** (**Empty Status Signal** is set); therefore there cannot be made any read operations. If SR = Buffer Size, then the buffer is **full** (**Full Status Signal** is set); therefore no more write operations are allowed until SR is decremented. The Empty Status Signal informs the **packet forwarding unit (PFU)** in the current module (NI or Routers) about the availability of a valid packet for a read operation. The Full Status Signal generates **Accept Signal**, which informs the previous module's (NI or Router) PFU about the availability of buffer space.

We implement two variations of the input buffer in our design. The first variation is used in the Input Network Interface and stores ATM packets. The second variation is used in the Routers and Output Network Interface, storing U(M)DN flits. The details of these input buffers are explained in the following sections.

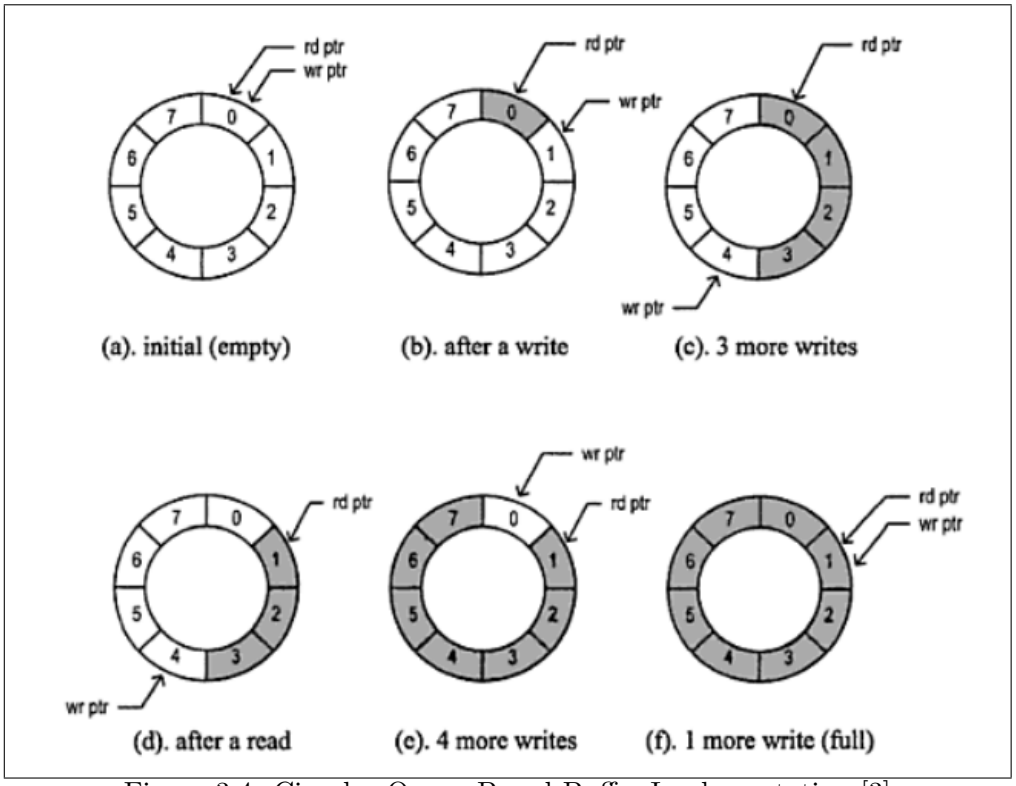


Figure 3.4: Circular Queue Based Buffer Implementation [3]

3.4 Network Interface (NI)

Network Interface (NI) is the module that acts as an abstraction layer in between the network protocol and UDN/MDN switch protocols. There are two types of NI: **Input NI (INI)** and **Output NI (ONI)**. When an ATM packet is injected into the switch, INI encapsulates (packetizes) the ATM packets into U(M)DN packets, and transmits them to the next router, in equally sized flits. ONI, in return, receives the U(M)DN flits from the last router in a packet's routing path, strips (depacketizes) the ATM packet from U(M)DN packet and ejects it from the switch. In the next subsections, we explain the details of INI and ONI hardware design.

3.4.1 Input Network Interface

When an ATM Packet is injected into the crossbar switch fabric from a line card, the first module that it will be processed is the **Input Network Interface (INI)**. The three tasks of the INI are,

- Buffering the ATM Packets injected into the switch from the input line cards,

- Packetizing an ATM Packet into U(M)DN Packet,
- Transmitting the U(M)DN packet, divided into flits, to the next router.

The block diagram of **INI** and **INI Input Buffer** are presented in Figure 3.5 and Figure 3.6, respectively. The step-by-step packet flow is as follows: If INI Input Buffer is not full (Accept Out = 1), then the line card sets Write Enable signal (WrEn = 1) and forwards an ATM packet to INI **ATM in** port, in the same cycle. It is the **Packetizer**, where the ATM packet is encapsulated into U(M)DN payload, and the routing information in the U(M)DN header is generated, as explained in Section 3.2. **INI Packet Forwarding Unit (IPFU)** is composed of a **flit counter**, and a **finite state machine (FSM)**, with **(IDLE, READ)** states. If the input buffer on the next router is available (Accept In = 1) and INI Input buffer is not empty (FIFO Empty = 0), FSM makes a transition from IDLE to READ state; IPFU issues a read operation (RdEn = 1) from the current input buffer, and a write operation to the next router's input buffer (WrEn Out = 1); and the flit counter is incremented each cycle to index (select) the individual flits of the U(M)DN packet, as they are forwarded to the next router, from the **Flit Out** port. Once the whole packet is forwarded, FSM makes a transition to IDLE state, and waits for the conditions (Accept In = 1) and (FIFO Empty = 0) to be met in order to issue another forwarding operation.

The flow in the INI Input Buffer is exactly as explained in Section 3.3. Accept Out signal is equal to the inverse of Full Status Signal.

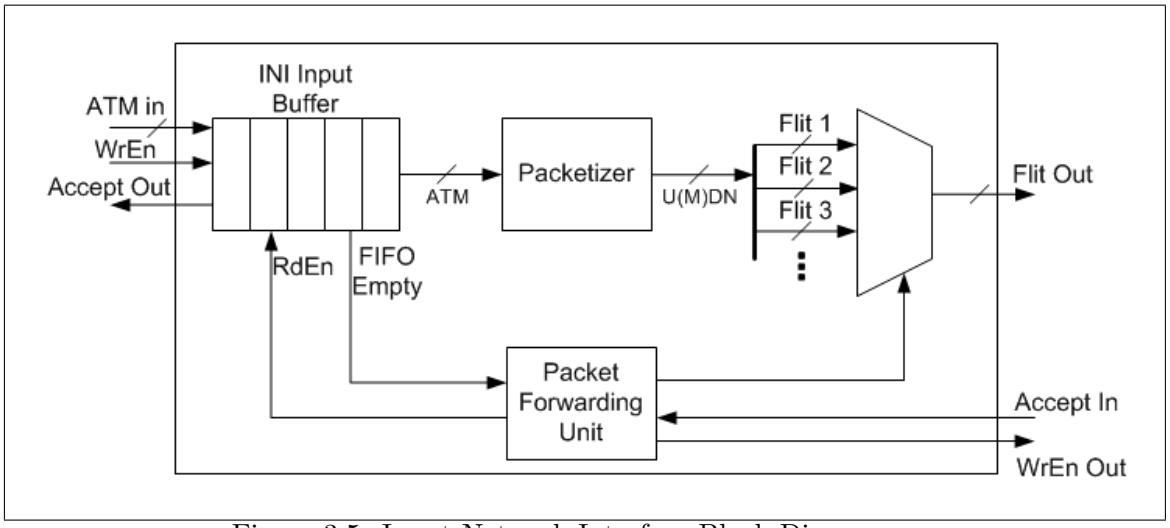


Figure 3.5: Input Network Interface Block Diagram

3.4.2 Output Network Interface

The U(M)DN packet is transmitted through the routers in the switch, until it reaches the **Output Network Interface (ONI)** connected to the packet's destination output port,

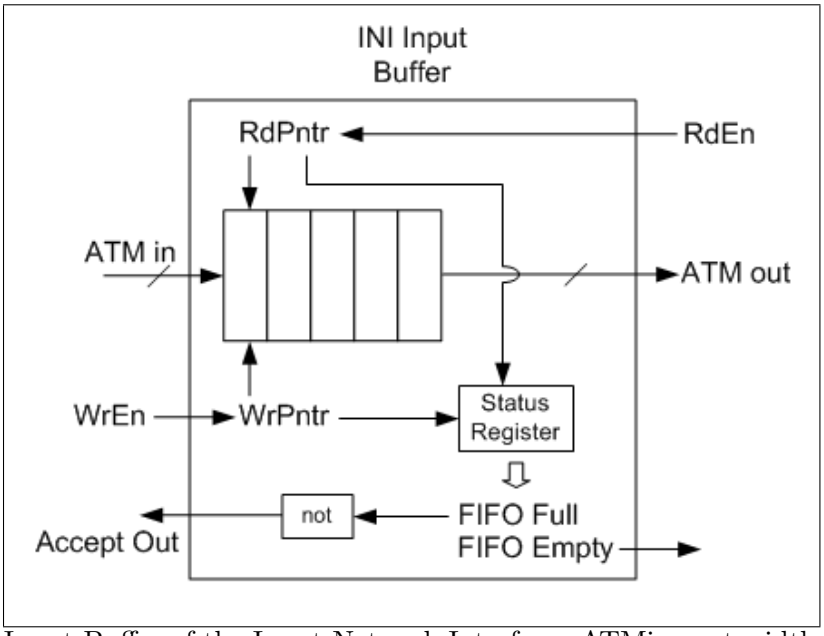


Figure 3.6: Input Buffer of the Input Network Interface: ATMin port width is 53-bytes, and FlitOut port width 9-bytes. PFU selects individual flits from the U(M)DN packet and serializes the transfer to the next routers.

where the original ATM packet is stripped (depacketized) from the U(M)DN packet. The three tasks of the ONI are,

- Buffering the U(M)DN Packets, divided into flits,
- De-packetizing an U(M)DN Packet into an ATM Packet,
- Ejecting the ATM Packet to the output line card.

The block diagram of **ONI** and **Router Input Buffer** are presented in Figure 3.7 and Figure 3.8, respectively. ONI employs the same input buffer module as the routers, because their input data type is Flit. The step-by-step packet flow is as follows: If INI Input Buffer is not full (Accept Out = 1), then the previous router's PFU sets Write Enable signal (WrEn = 1) and forwards an U(M)DN packet, divided into flits, to ONI Flits In port. It is the Depacketizer, where The ATM is stripped from the U(M)DN payload. The ONI Packet Forwarding Unit (OPFU) is composed of a finite state machine (FSM), with (IDLE, READ) states. If the output line card is available (Accept In = 1) and the input buffer is not empty (FIFO Empty = 0, and therefore it has at least one complete packet), the FSM makes a transition from IDLE to READ state; OPFU issues a read operation (RdEn = 1) to the input buffer, and a write operation to the output line card (WrEn Out = 1); and the ATM packet is forwarded to the output line card. Please note that OPFU is simpler than IPFU, since it does not embody a counter to index individual flits. Once the ATM packet is forwarded, FSM makes a transition

to IDLE state, and waits for the conditions (Accept In = 1) and (FIFO Empty = 0) to issue another forwarding operation.

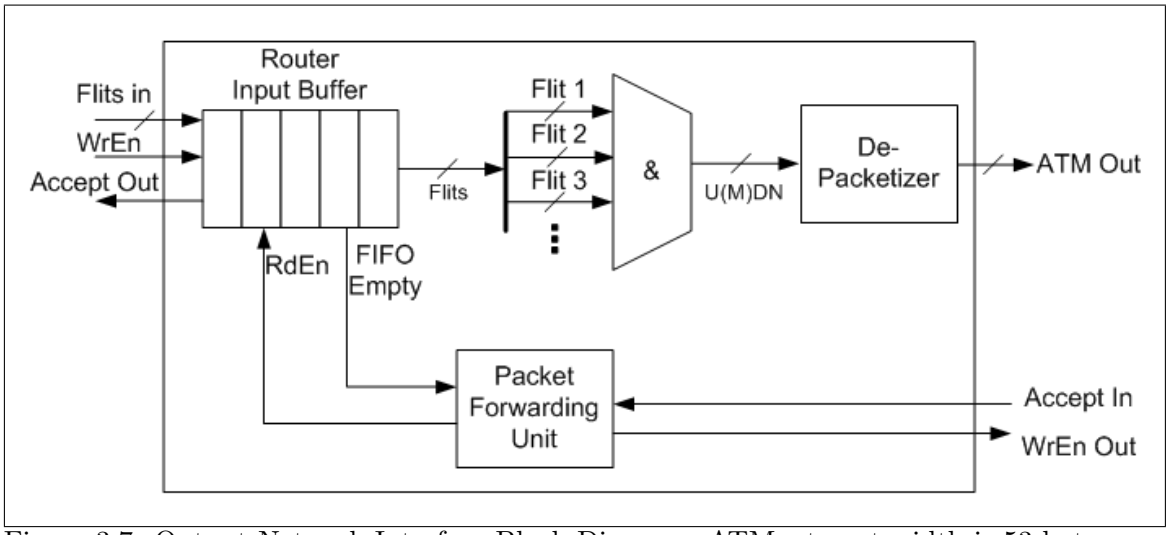


Figure 3.7: Output Network Interface Block Diagram: ATMout port width is 53-bytes, and FlitIn port width 9-bytes. The ATM packet is stripped from the U(M)DN Wrapper and ejected to the output line card.

Router (ONI) input buffer requires more complex control circuitry than the INI input buffer, as can be observed in Figure 3.8. It complies with the explanations in Section 3.3, and in addition it has a **Write Control Unit**, which is composed of a **flit counter**, and a **finite state machine (FSM)**, with (**IDLE, WRITE**) states. If the input buffer is available (Accept Out = 1), the previous router issues a write operation (WrEn = 1) and starts transmitting Flits, one at a cycle. FSM makes a transition from IDLE to WRITE state; and the flit counter is incremented each cycle to index (select) the flit sections of the register memory. Once the whole packet is accepted and written on the buffer, FSM makes a transition to IDLE state. Another packet is accepted only when the conditions (FSM = IDLE) and (FIFO Full = 0) are met.

3.5 UDN Router Design

The **northmost row routers** of UDN switch have two input (West, South) and two output (East, South) ports; similarly, the **southmost row routers** have two input (West, North) and two output (East, North) ports. The **central routers** have three input (West, South, North) and three output (East, South, North) ports. This implies that the **traffic flow** can occur only through seven input/output combinations in UDN central routers, as shown in Figure 3.9.

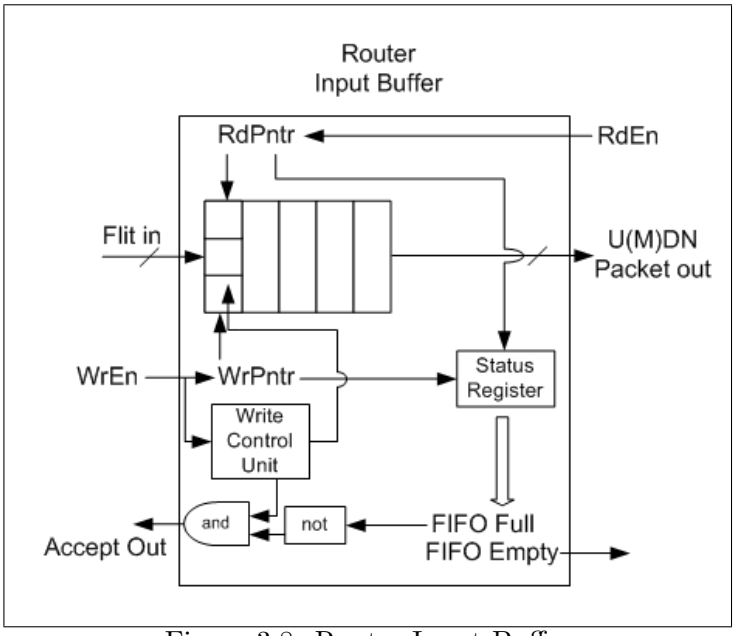


Figure 3.8: Router Input Buffer

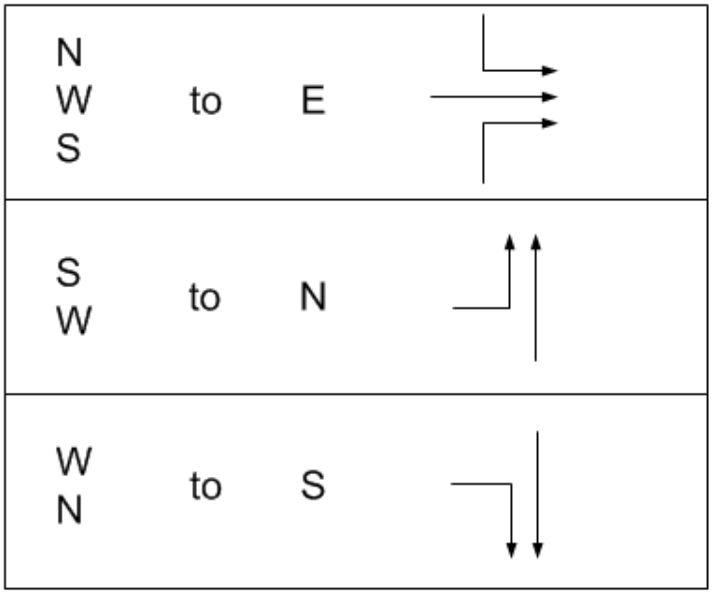


Figure 3.9: Possible Traffic Flows on UDN Switch (on 3 I/O Port Routers)

3.5.1 Router Types

It is important to investigate if the physical input/output port combinations exactly match the possible traffic flows. In this section, we present the findings of a simulator we have written in C to investigate which types of traffic flows occur on UDN switches of

any given (N, M) size, over UDN XY Modulo Routing Algorithm (discussed in Chapter 4).

In Figure 3.10, we present the results for the UDN switches of size (N, M) = (2, 1), (4, 3), (8, 7), (16, 15). The values corresponding to the southmost and northmost rows (0 and 7) imply 2 I/O Port Routers; whereas the central values (1 to 7) correspond to 3 I/O Port Routers with different traffic flows, for which the legend is provided in Figure 3.11. The simulations reveal that there are 8 types of routers, which are being repetitively used in UDN switches with any (N, M) size.

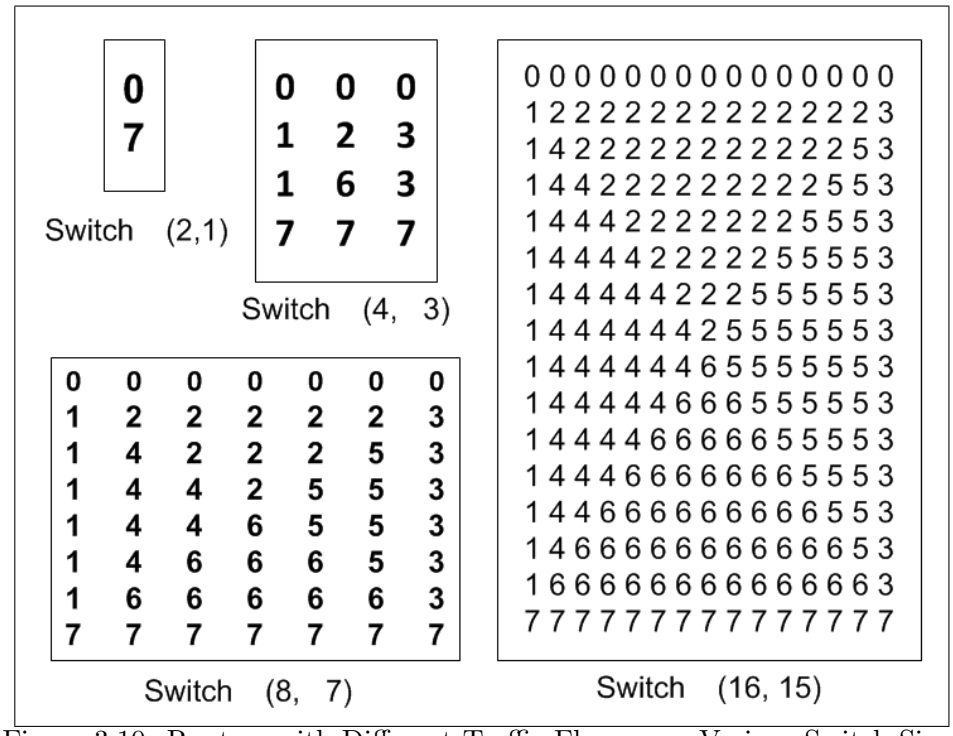


Figure 3.10: Routers with Different Traffic Flows over Various Switch Sizes

These router types do not only have different traffic flows, but also different frequencies on these traffic flows. In other words, the frequency of a packet to travel through a distinct I/O port pair in a router depends on the router type. This observation will be useful in the scheduling related section, to explain the weighted arbitration. We first investigate the UDN switch and, then discuss shortly why this does not apply to MDN switch.

In Figure 3.12, a UDN switch with (N, M) = (3, 2) is presented, with its traffic flow map. Each router is named with a capital letter, and I/O ports of the switch are named i1, o1, etc. Figure 3.13 shows the routing paths for all I/O port pairs, over UDN XY Modulo routing algorithm, with the T-Values set to 0.

When a packet is transmitted from an input port to an output port, it travels through certain routers, as listed in Table 3.1. We also tabulate the I/O port pairs on the routing

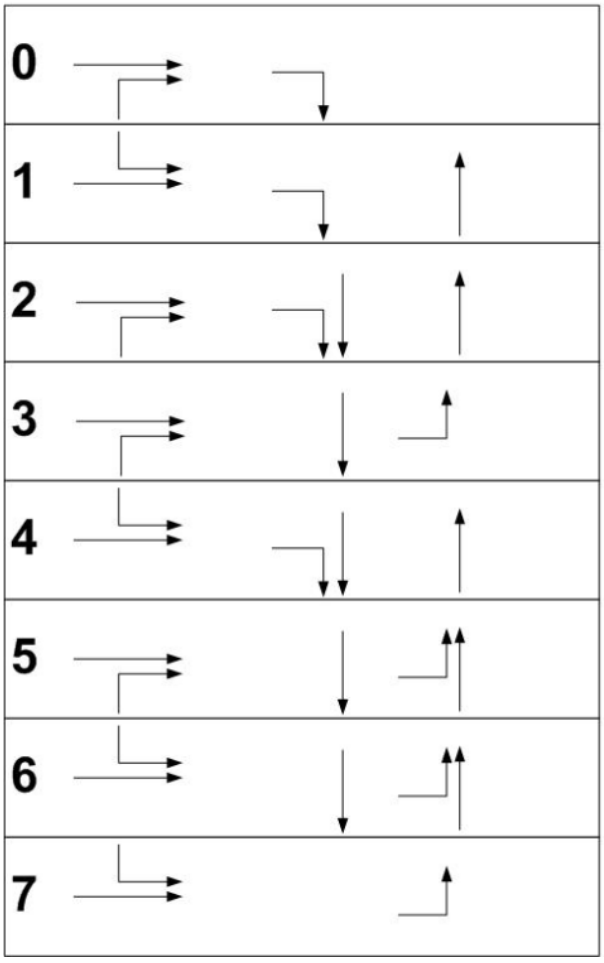


Figure 3.11: Legend for the Traffic Flows

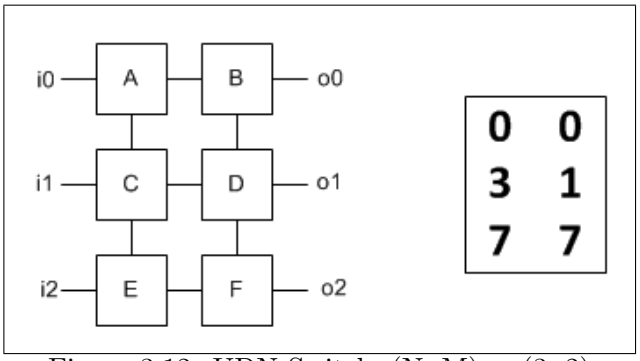


Figure 3.12: UDN Switch, (N, M) = (3, 2)

paths, in Table 3.2. For instance, if a packet is transmitted from i0 to o1, then this will be shown as A (W → E), B (W → S), D (N → E). Finally, we also tabulate the flows

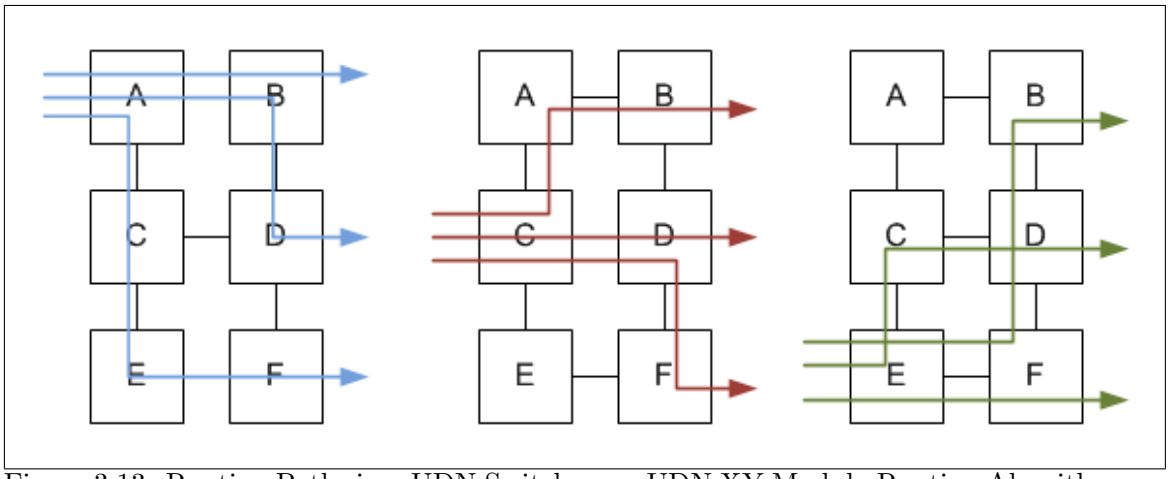


Figure 3.13: Routing Paths in a UDN Switch, over UDN XY Modulo Routing Algorithm

Table 3.1: List of Routers in the Routing Path of the I/O Port Pairs

I0 → O0 : A, B
I0 → O1 : A, B, D
I0 → O2 : A, C, E, F
I1 → O0 : C, A, B
I1 → O1 : C, D
I1 → O2 : C, D, F
I2 → O0 : E, F, D, B
I2 → O1 : E, C, D
I2 → O2 : E, F

on individual routers, in Table 3.3.

On Table 3.4, it can be observed that certain flows through some I/O port pairs are

Table 3.2: List of Port Pairs on the path in between the Switch's I/O Pairs

I0 → O0 : A (W → E), B (W → E)
I0 → O1 : A (W → E), B (W → S), D (N → E)
I0 → O2 : A (W → S), C (N → S), E (N → E), F (W → E)
I1 → O0 : C (W → N), A (S → E), B (W → E)
I1 → O1 : C (W → E), D (W → E)
I1 → O2 : C (W → E), D (W → S), F (N → E)
I2 → O0 : E (W → E), F (W → N), D (S → N), B (S → E)
I2 → O1 : E (W → N), C (S → E), D (W → E)
I2 → O2 : E (W → E), F (W → E)

Table 3.3: Traffic Flows on Individual Routers

A : (W → E), (W → E), (W → S), (S → E)
B : (W → E), (W → E), (W → S), (S → E)
C : (W → E), (W → E), (S → E), (N → S), (W → N)
D : (W → E), (W → E), (N → E), (S → N), D (W → S)
E : (W → E), (W → E), (N → E), (W → N)
F : (W → E), (W → E), (N → E), (W → N)

Table 3.4: Frequency of Traffic Flows on Router Types

		Router Types			
Arbiter	**I/O Port Pairs**	**0**	**3**	**1**	**7**
East	**W → E**	2	2	2	2
	N → E		0	1	1
	S → E	1	1	0	
North	**W → N**		1	0	1
	S → N		0	1	
South	**W → S**	1	0	1	
	N → S		1	0	

more frequent than the others. For example, in Router A, (W → E) traffic is twice as frequent as (W → S) or (S → E). Another observation is that the same router types on a switch have the same frequencies for the traffic flows; A and B are Type 0, E and F are type 7 and have the same frequencies.

It is also noteworthy to observe that when N = M in a UDN switch (Figure 3.14) and T-Value is set to 0, there is an abundant column of routers, without any active routing (i.e. only West → East flow); in this case, the optimal switch would be M = N 1.

However, with two types of 2 I/O Port and six types of 3 I/O Port UDN Routers, the switch implementation quickly gets complicated, because:

- Any changes made on the RTL description for one module needs to be made on all the others,
- Reusability and scalability of the design suffers from the large number of router types,
- Preprocessor constructs of hardware definition languages (i.e. VHDL) are very weak, and therefore make it problematic to instantiate the different types of routers on the corresponding (Row, Column) index of the crossbar switch.

Because of these reasons, some reduction of the router types is needed, **in exchange**

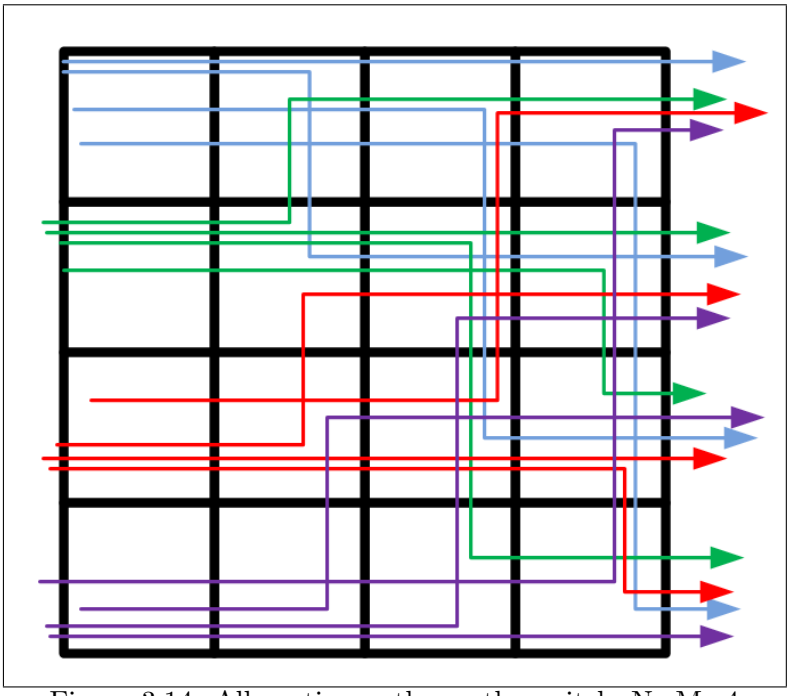


Figure 3.14: All routing paths on the switch, N=M=4

of some decrease in the performance, discussed further in Sections 4.2.1 and 6.4. On Figure 3.11, it can be observed that router types 0 and 7 are the mirrored copies of each other. Using a generic position constant (North or South), it is possible to differentiate these two types. All the other routers, 1 to 7, have 3 I/O ports, and can also be reduced to one single module; in this case, the traffic flow differences need to be resolved within the arbitration unit and scheduling algorithm, by eliminating the I/O combinations for the non-existent traffic flow (discussed in Section 4.2.1).

3.5.2 UDN Router Architecture

Hardware implementation of the 2 and 3 I/O Port UDN Routers are very similar, because they use the same input buffers and Packet Forwarding Units (PFU). The arbiters are also based on the same principles; however they get more complex as the number of I/O ports increase. In this section we describe the overall architecture of the UDN router and discuss the routing and arbitration in Chapter 4.

The block diagram for a 3 I/O Port Router is given in Figure 3.16. There are 3 input ports, West, North and South; and there are 3 output ports, East, North and South. The router input buffer modules are placed at each input port, whereas the PFUs are placed to each output port. Routers use the same IPFU modules as INIs.

Here is a step-by-step flow example, to illustrate how a packet is forwarded through a router. In this example, we look from the perspective of IPFU at the East Output Port.

All three of the West, South and North input ports will compete for the East Output Port, if

- All three input ports have at least one packet in their input buffers (Empty Signal = 0), and
- The routing unit computes that the head-of-line packets in all input ports are to be ejected over the East Output Port.

However, with three input ports competing for the same output port, there is contention. The contention is handled by the scheduling algorithm, which controls the arbiter to "virtually connect" one of the input ports to the East output port. In this way, IPFU can perform the read operations from the respective input buffer. This is illustrated in Figure 3.15, where the red line (thick horizontal line in B&W copy) symbolizes the arbiter. Read Enable (RdEn), FIFO Empty and Flit Indexing signals are interfaced through the arbiter, in between the IPFUs and input buffers. Further details on arbitration and scheduling algorithms are discussed in Chapter 4.

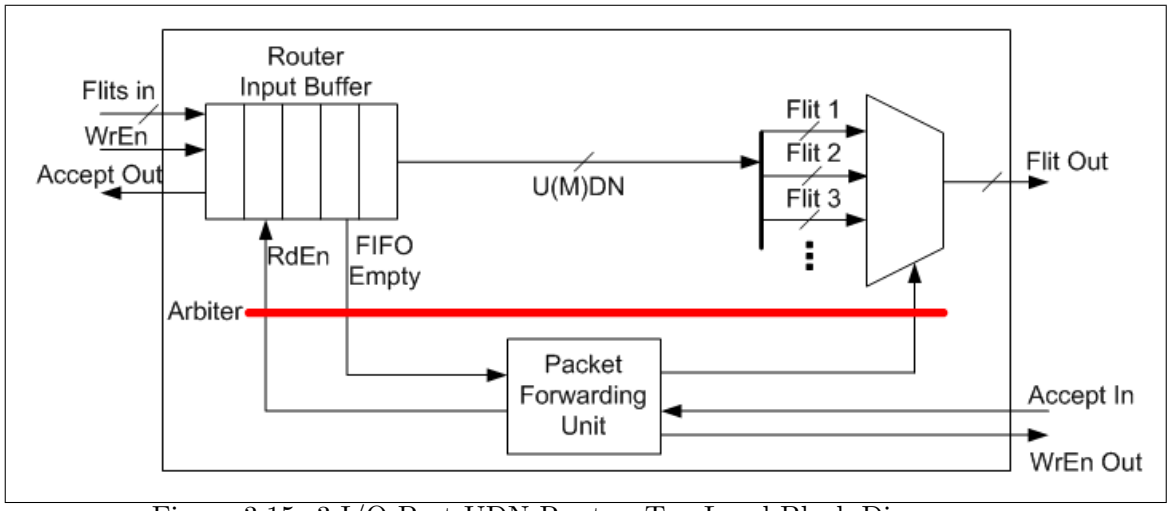


Figure 3.15: 3 I/O Port UDN Router, Top-Level Block Diagram

3.6 Instantiating Routers and NIs in the UDN Switch

The Routers and NIs are instantiated using VHDL preprocessor command ***generate***, such that many design parameters of the switch, like switch size, buffer width, U(M)DN packet size, etc. can be controlled with constant values. In this way, the changes on the VHDL code are easily made to adjust the switch parameters.

The connections in between the router ports and NIs have to be addressed according to a certain indexing scheme, such that router signals can be connected correctly

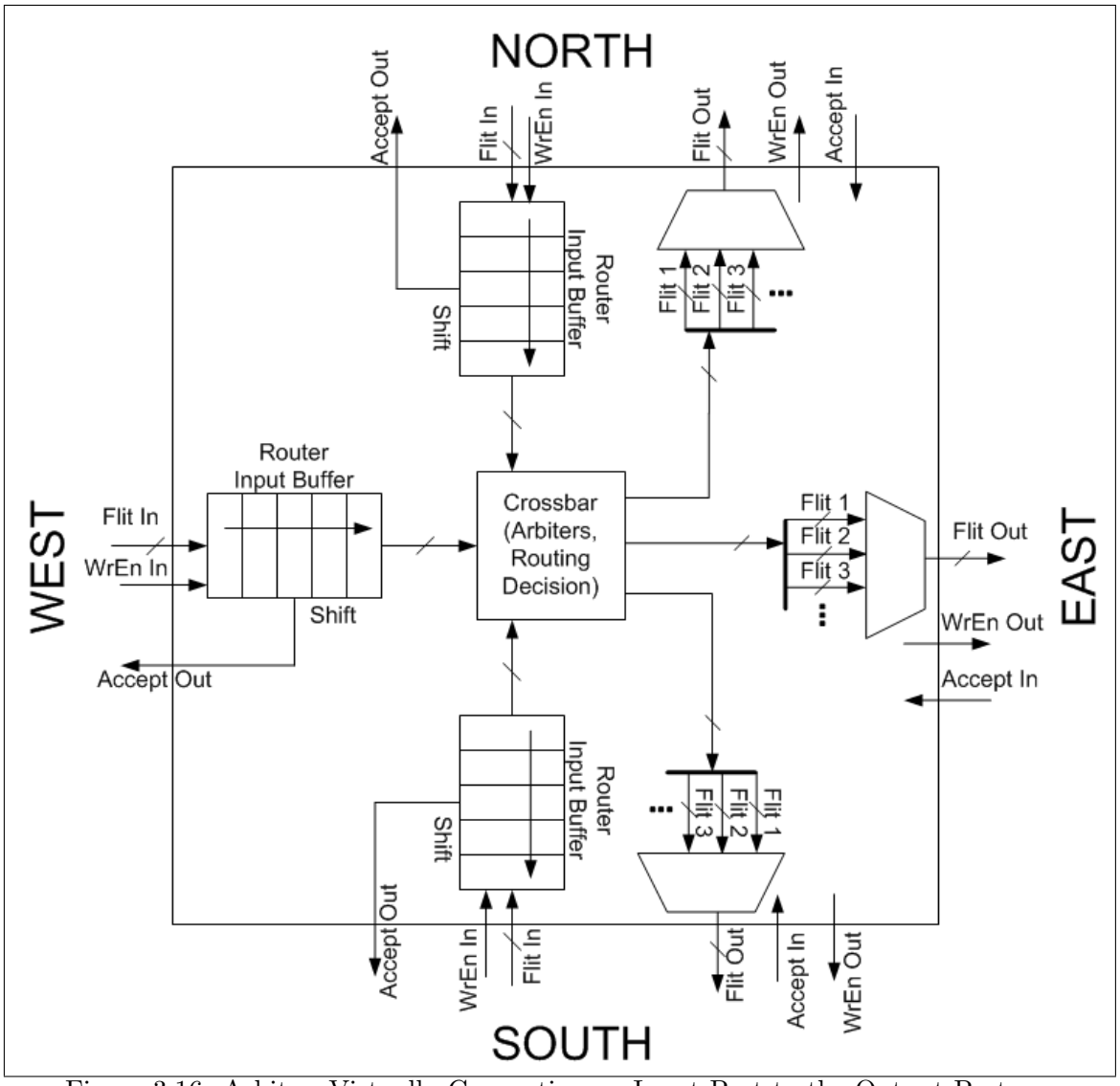


Figure 3.16: Arbiter, Virtually Connecting an Input Port to the Output Port

under the *generate* code blocks. In the UDN Routers, all input and output ports have inbound and outbound flow control signals. These signals should be connected to the corresponding signals in the previous or following routers. We define the 3-tuple **UDN Switch Index** as Signal Name (**Source Output Port, Destination Column Address, Destination Row Address**).

In order to describe the UDN Switch Index better, we present an example UDN switch of size (2, 2), in Figure 3.17. The Routers and NIs are indexed together with (Column Address, Row Address) indices. The East output port of INI(0, 0) is connected to the West input port of Router(1, 0), which is composed of the inbound Write Enable and Flits In signals and outbound Accept Out Signals. Write Enable signal, is indexed as

WE(E, 1, 0), because Source Output Port is East and the destination module is indexed as (1, 0).

The code segment in Listing 3.1 instantiates the matrix of central UDN Routers with 3 I/O ports. In this way, the respective signals are connected in between the corresponding router and NI modules.

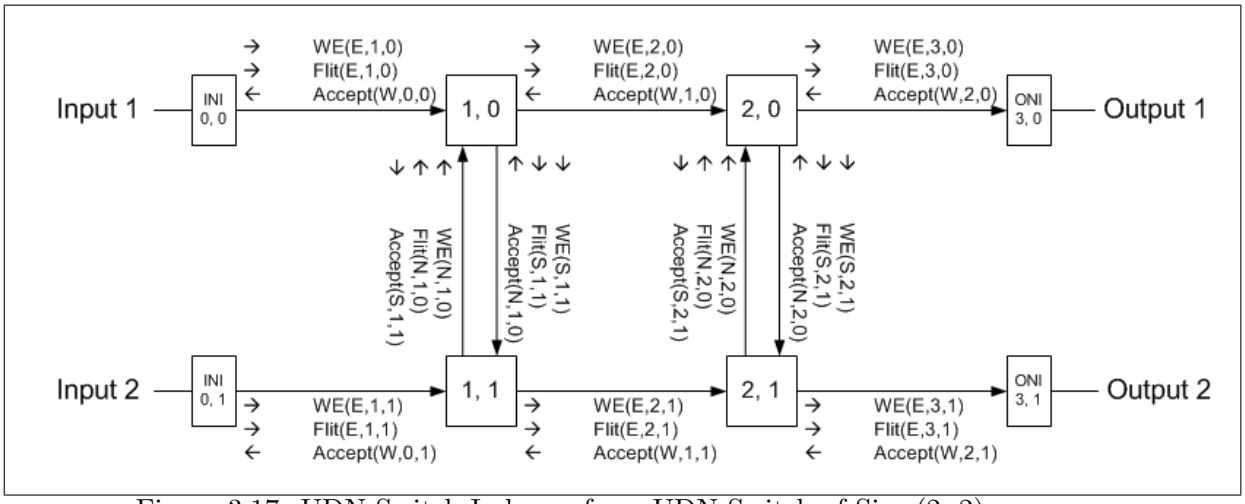


Figure 3.17: UDN Switch Indexes, for a UDN Switch of Size (2, 2)

Listing 3.1: Nested Generate for 3 I/O Port Router Instantiation

```
Gen_Router_cR: for i in 1 to SwitchN-2 generate          --Row(i)
    Gen_Router_cC: for j in 0 to SwitchM-1 generate      --Column(j)
        Router_c : Router3UDN
        generic map
        (
            RouterColAddr   => j+1,
            RouterRowAddr   => i
        )
        port map
        (
            reset           => reset,
            clk             => clk,
            --
            acceptOutW      => internalAccepts(WEST)(j)(i),
            UDN_flit_inW    => internalFlits(EAST)(j+1)(i),
            writeEn_inW     => internalWE(EAST)(j+1)(i),
            ---
            acceptOutN      => internalAccepts(NORTH)(j+1)(i-1),
            UDN_flit_inN    => internalFlits(SOUTH)(j+1)(i),
            writeEn_inN     => internalWE(SOUTH)(j+1)(i),
            ---
            acceptOutS      => internalAccepts(SOUTH)(j+1)(i+1),
            UDN_flit_inS    => internalFlits(NORTH)(j+1)(i),
            writeEn_inS     => internalWE(NORTH)(j+1)(i),
            ---
            acceptInE       => internalAccepts(WEST)(j+1)(i),
            UDN_flit_outE   => internalFlits(EAST)(j+2)(i),
            writeEn_outE    => internalWE(EAST)(j+2)(i),
            ---
            acceptInN       => internalAccepts(SOUTH)(j+1)(i),
```

```
            UDN_flit_outN   => internalFlits(NORTH)(j+1)(i-1),
            writeEn_outN    => internalWE(NORTH)(j+1)(i-1),
            ---
            acceptInS       => internalAccepts(NORTH)(j+1)(i),
            UDN_flit_outS   => internalFlits(SOUTH)(j+1)(i+1),
            writeEn_outS    => internalWE(SOUTH)(j+1)(i+1)
        );
    end generate;
end generate;
```

3.7 MDN Router Design

The MDN architecture is based on the UDN switch; therefore, most of the UDN modules are also being used in the MDN switch, including INI, ONI, IPFU, INI Input Buffer, and Router Input Buffer. The arbiter is also very similar in terms of basic principles.

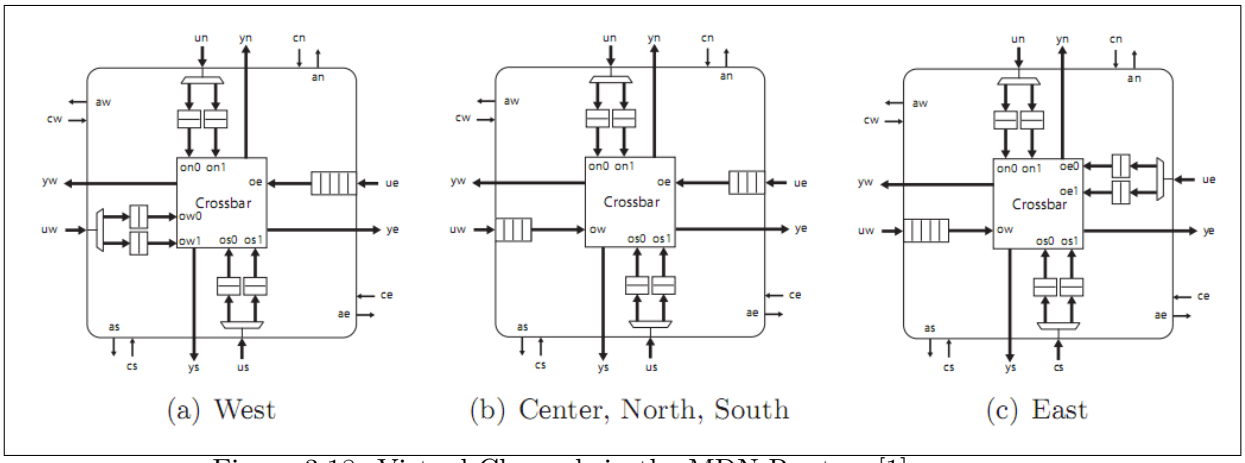
On the other hand, the MDN switch requires 4 I/O Port Routers, because the MDN switch architecture has the input/output pins placed all around the layout peripheral. Due to this fact, there is only one type of MDN Router, which permits traffic flow in all sixteen possible I/O combinations, with the same frequency. The MDN Router has four Router Input Buffers at the input ports, four IPFU at the output ports and a more complex arbiter, compared to the 2 and 3 I/O Port UDN Routers due to the increase in number of traffic flows.

3.8 MDN Virtual Channels

MDN switch architecture is prone to deadlocks, because of its multidirectional nature. In [30], it was proposed that virtual channels can be used to avoid deadlocks. [1] proposes a functional-level design for virtual channels, to remedy the deadlock problem in MDN switch. According to the proposal, two virtual channels (VC0, VC1) are used in North and South input ports. In addition, the West input port of the westmost column routers and the East input port of the eastmost column routers are virtual channeled. If the destination of a packet is the east of the current router, then VC0 is used, whereas in the other cases VC1 is used. Figure 3.18 represents the block diagrams of the three MDN Routers with virtual channels at different input ports.

In [1], the packets at a virtual channeled input port are demultiplexed into the virtual channels; however, the virtual channel outputs are directly connected to the arbiters, without any multiplexing, resulting in a much more complex arbiter due to the increased number of resources competing for the output ports. Our proposal involves the use of a multiplexing unit, in order to implement the virtual channels on one physical link, such that the flow control will perceive it just like any other input buffer (Figure 3.19).

Two Router Input Buffers are instantiated in the Virtual Channel, without any changes. The flow control and flit input/output signals are connected to the Virtual



(a) West (b) Center, North, South (c) East

Figure 3.18: Virtual Channels in the MDN Routers [1]

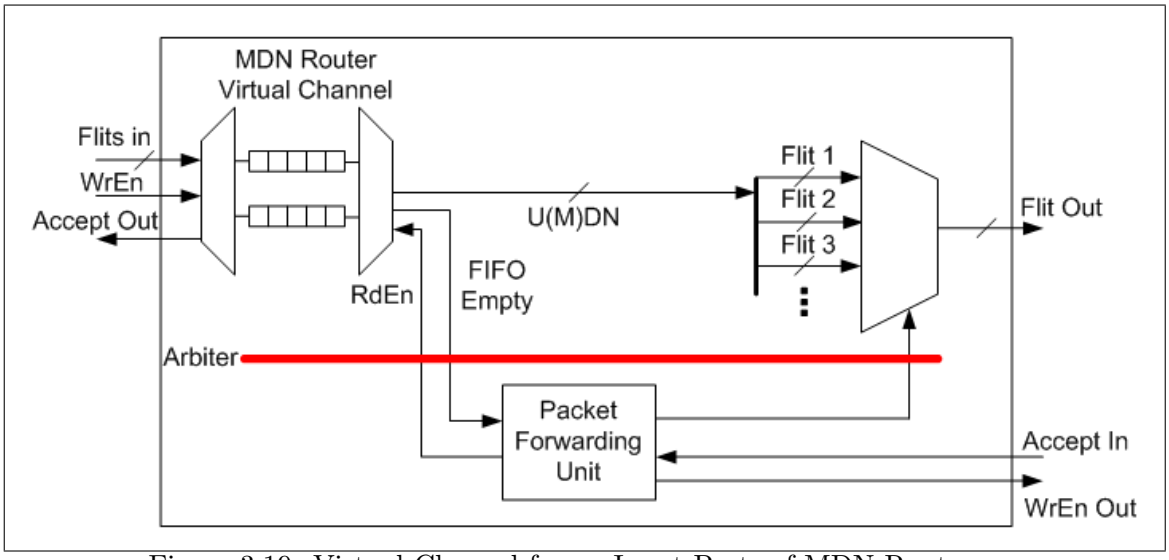


Figure 3.19: Virtual Channel for an Input Ports of MDN Router

Channel interface with multiplexing and demultiplexing circuitry. The first flit of the packet has the destination field; we need to compute if the packet is destined to an output address to the east of the current router, and demultiplex the packet and Write Enable signal accordingly.

We implement a small arbitration unit, to select one of the two channels competing for the virtual channel output port. The arbitration is based on Round Robin scheduling, where the channels are selected in order, with fair time sharing. The channel is changed, when the PFU issues a Read Enable signal; however if a read enable signal is not issued over a certain time (**timeout period**), then the channel is changed automatically. The signals U(M)DN and FIFO Empty are multiplexed and the Read Enable signal is demultiplexed according to the RR value.

Our VHDL implementation instantiates a Router Input Buffer or Virtual Channel according to the position of the MDN Router in the MDN switch, using the generate command; thus, we do not have multiple VHDL codes for the MDN Routers.

3.9 Instantiating Routers and NIs in the MDN Switch

Like the UDN switch, MDN switch also requires an indexing scheme for module instantiation. Similar to the UDN Switch Index, we define the MDN Switch Index, as Signal Name (Source Output Port, Destination Column Address, Destination Row Address) (Figure 3.21).

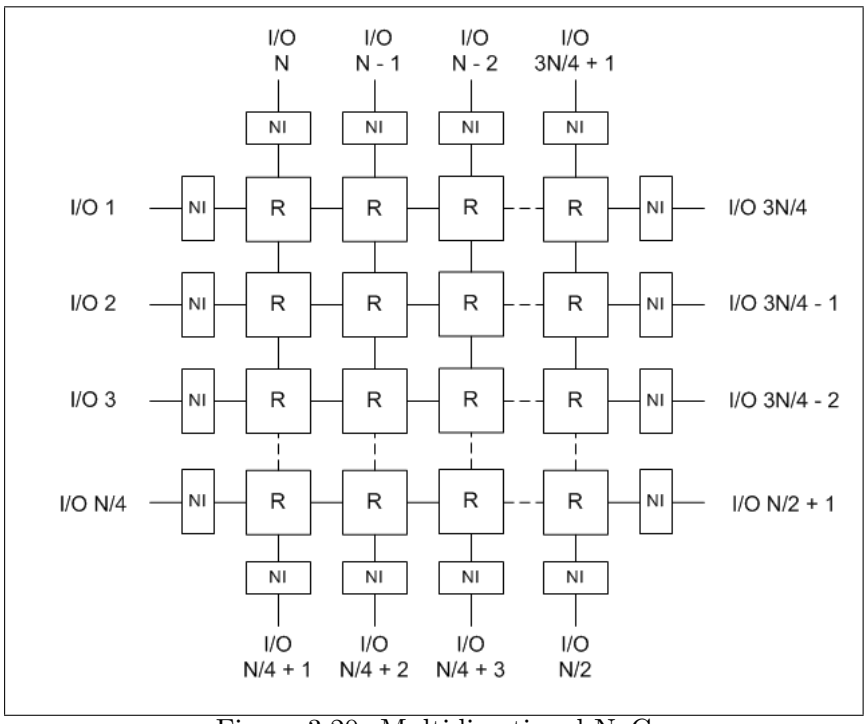


Figure 3.20: Multidirectional NoC

In UDN, all input/output ports and NIs were on the West and East sides respectively, making it easy to describe which NIs should be connected to which router. In MDN, it is additionally required to determine on which side of the MDN switch is a certain NI. We have implemented two VHDL functions to convert an I/O port address into MDN switch index. The first function (NIconnIN, Listing 3.2) is used for determining the MDN switch index for inbound signals, whereas the second (NIconnOUT, Listing 3.3) is for the outbound signals (for both INI and ONI). The functions basically check if the value of an I/O port address is smaller than the corner values of the MDN switch (e.g. Southwest corner = N/4 + 1), to compute on which side of the layout the port is (Figure 3.20). MdnSwitchDim constant is equal to N/4, defining number of I/O port pairs per switch side. These functions are called whenever an NI is instantiated in the

top-level MDN switch design, to compute the MDN switch Index of NI signals, such that they are correctly connected to the corresponding routers.

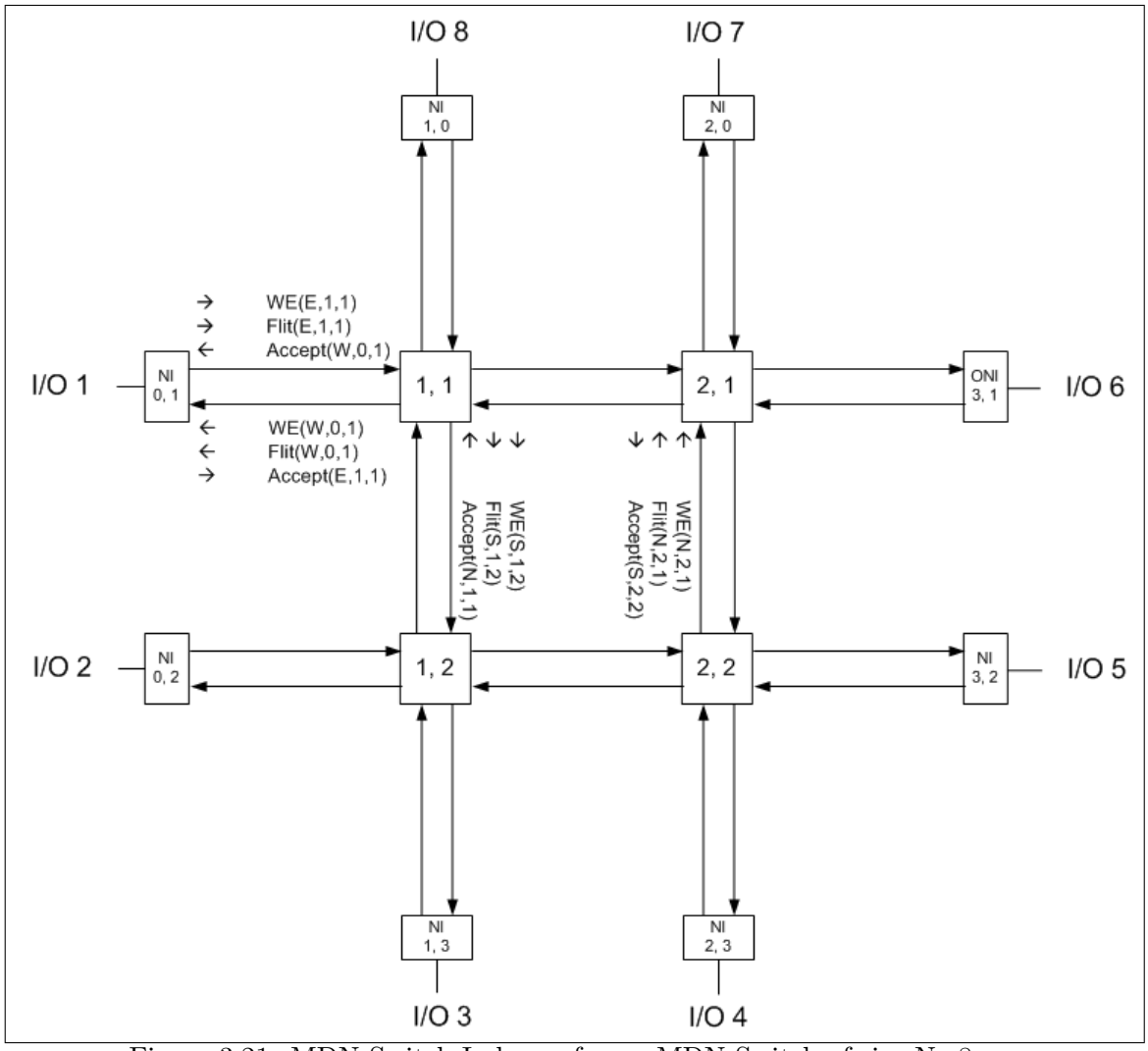


Figure 3.21: MDN Switch Indexes, for an MDN Switch of size N=8

Listing 3.2: Converting the Switch Output Address to MDN Switch Index for INI

```
function NIconnIN (destAddr : integer range 0 to (SwitchN-1)) return NIconnection is
    variable NIconnect : NIconnection;
begin
    if (destAddr < (SwitchN/4 + 1)) then
        NIconnect.dir   := WEST;
        NIconnect.row   := conv_integer(destAddr)+1;
        NIconnect.col   := 0;
    elsif (destAddr < (SwitchN/2 + 1)) then
        NIconnect.dir   := SOUTH;
        NIconnect.row   := MdnSwitchDim+1;
        NIconnect.col   := conv_integer(destAddr)-MdnSwitchDim+1;
    elsif (destAddr < ( SwitchN*3/4 + 1)) then
        NIconnect.dir   := EAST;
```

```
        NIconnect.row   := (3*MdnSwitchDim)-conv_integer(destAddr);
        NIconnect.col   := MdnSwitchDim+1;
    else
        NIconnect.dir   := NORTH;
        NIconnect.row   := 0;
        NIconnect.col   := MdnSwitchM-conv_integer(destAddr);
    end if;
    return NIconnect;
end NIconnIN;
```

Listing 3.3: Converting the Switch Output Address to MDN Switch Index for ONI

```
function NIconnOUT (destAddr : integer range 0 to (SwitchN-1)) return NIconnection is
    variable NIconnect  : NIconnection;
begin
    if (destAddr < (SwitchN/4 + 1)) then
        NIconnect.dir   := EAST;
        NIconnect.row   := conv_integer(destAddr)+1;
        NIconnect.col   := 1;
    elsif (destAddr < (SwitchN/2 + 1)) then
        NIconnect.dir   := NORTH;
        NIconnect.row   := MdnSwitchDim;
        NIconnect.col   := conv_integer(destAddr)-MdnSwitchDim+1;
    elsif (destAddr < (SwitchN*3/4 + 1)) then
        NIconnect.dir   := WEST;
        NIconnect.row   := (3*MdnSwitchDim)-conv_integer(destAddr);
        NIconnect.col   := MdnSwitchDim;
    else
        NIconnect.dir   := SOUTH;
        NIconnect.row   := 1;
        NIconnect.col   := SwitchN - conv_integer(destAddr);
    end if;
    return NIconnect;
end NIconnOUT;
```

Listing 3.4: Calling Listing 3.2 and Listing 3.3 to Instantiate Input NIs on MDN Switch

```
Gen_NIinputs: for i in 0 to SwitchN-1 generate
    NIinputs : NIinput
    generic map
    (
        NIaddr      => i
    )
    port map
    (
        reset       => reset,
        clk         => clk,
        ---
        ATMin       => SwitchATMin(i),
        WEin        => SwitchWEin(i),
        acceptOut   => SwitchAcceptOut(i),
        ---
        acceptIn    => internalAccepts(NIconnIN(i).dir)(NIconnIN(i).col)(NIconnIN(i).row),
        flit_out    => internalFlits(NIconnOUT(i).dir)(NIconnOUT(i).col)(NIconnOUT(i).row),
        WEout       => internalWE(NIconnOUT(i).dir)(NIconnOUT(i).col)(NIconnOUT(i).row)
    );
end generate;
```

The code segment below (Listing 3.5) instantiates the matrix of MDN Routers. In this way, all signals are connected in between the corresponding routers and NI modules.

Listing 3.5: Nested Generate Loops for MDN Router Matrix Instantiation

```
Gen_Router_MDN_R: for i in 1 to MdnSwitchDim generate        --Row(i)
    Gen_Router_MDN_C: for j in 1 to MdnSwitchDim generate    --Column(j)
        Router4_MDN : Router4MDN
        generic map -- Deleted for saving from the space in the report.
        port map
        (
            reset           => reset,
            clk             => clk,
            --
            acceptOutW      => internalAccepts(WEST)(j-1)(i),
            UDN_flit_inW    => internalFlits(EAST)(j)(i),
            writeEn_inW     => internalWE(EAST)(j)(i),
            ---
            acceptOutE      => internalAccepts(EAST)(j+1)(i),
            UDN_flit_inE    => internalFlits(WEST)(j)(i),
            writeEn_inE     => internalWE(WEST)(j)(i),
            ---
            acceptOutN      => internalAccepts(NORTH)(j)(i-1),
            UDN_flit_inN    => internalFlits(SOUTH)(j)(i),
            writeEn_inN     => internalWE(SOUTH)(j)(i),
            ---
            acceptOutS      => internalAccepts(SOUTH)(j)(i+1),
            UDN_flit_inS    => internalFlits(NORTH)(j)(i),
            writeEn_inS     => internalWE(NORTH)(j)(i),
            ---
            acceptInW       => internalAccepts(EAST)(j)(i),
            UDN_flit_outW   => internalFlits(WEST)(j-1)(i),
            writeEn_outW    => internalWE(WEST)(j-1)(i),
            ---
            acceptInE       => internalAccepts(WEST)(j)(i),
            UDN_flit_outE   => internalFlits(EAST)(j+1)(i),
            writeEn_outE    => internalWE(EAST)(j+1)(i),
            ---
            acceptInN       => internalAccepts(SOUTH)(j)(i),
            UDN_flit_outN   => internalFlits(NORTH)(j)(i-1),
            writeEn_outN    => internalWE(NORTH)(j)(i-1),
            ---


            acceptInS       => internalAccepts(NORTH)(j)(i),
            UDN_flit_outS   => internalFlits(SOUTH)(j)(i+1),
            writeEn_outS    => internalWE(SOUTH)(j)(i+1)
        );
    end generate;
end generate;
```

UDN/MDN Routing & Scheduling

4

In Chapter 3, there have been made many forward references to this chapter, because the routing and scheduling units constitute the brain of the UDN and MDN switches. The routing unit computes the path a packet should travel in between the input and output ports of the switch, whereas the scheduling unit resolves the contention among the input buffers competing for the same output port and controls the arbiter to virtually connect one of the input buffers to the output port.

In this chapter, we propose routing and scheduling algorithms for UDN and MDN switches. [1] proposed the UDN XY Modulo and MDN XY Modulo routing algorithms. With some minor corrections and changes for better hardware performance, we apply UDN XY Modulo as it was proposed. MDN XY Modulo algorithm in [1], on the other hand, requires different input parameters than the UDN XY Modulo algorithm. This results in distinct UDM and MDN packets, with different header data fields. Instead, we propose a new algorithm for MDN routing, which uses the same input parameters as the UDN XY Modulo algorithm.

In [1], the scheduling scheme is not explained in details, but it is mentioned that it is Round Robin based. In this chapter, we also discuss some possible scheduling schemes based on Round Robin for UDN and MDN switches.

4.1 Routing in UDN and MDN Switches

In a NoC, the routing path for a packet can be computed all-at-once, i.e. at NI, as it is implemented in Best Effort Switch of Aetherial NoC [36] or incrementally, i.e. at the Routers. If all-at-once routing is chosen, this will create overhead in the U(M)DN packet, since the decision for each router has to be encoded on the packet header. On the contrary, in the incremental routing, the packet header does not have the full routing path, but the destination address; therefore, the routing path is computed one step at a time at each router, until the packet is ejected from the switch.

We choose to implement the incremental routing, because the overhead for all-at-once routing in the U(M)DN packet gets very high as the switch size increases and the header/payload ratio becomes inefficient. Here is a step by step explanation why we cannot implement a feasible all-at-once routing scheme for UDN switch. In a U(M)DN

packet,

Routing Path Byte-Width = Network Diameter * Bit-Width for Direction Encoding,
Network Diameter = (2 * CEILING (SQRT (N*M)) - 1
Bit-Width for Direction Encoding = LOG2 (Number of directions)
Number of Directions = 4 (i.e. West, North, South, East)

In a UDN switch of N=M=64, Routing Path = 127 × 2 bits ≈ 32 bytes, which is an unacceptable overhead with respect to 53-bytes payload encapsulating the ATM packet. In addition, all-at-once routing will only permit Unicast traffic, but will not be able to support Multicast.

Both UDN and MDN Routing Algorithms are based on modulo operation. The algorithms make a balanced distribution of the traffic over the columns or rows, thus earning its name XY Modulo; this means that the modulo operation is applied if the communication is on the X axis (from switch's West Input Port to switch's East Output Port as in UDN and MDN, or from switch East Input Port to switch West Output Port as in MDN) or on the Y axis (from switch's North Input Port to switch's South Output Port, or vice versa, as in MDN). In Figure 4.1, we exemplify the XY Modulo routing on the X axis, where packets are injected into the switch from i0 input port, with destinations to the o0 - o3 output ports. The packets are routed on different router columns per input/output port pairs, distributing the load on the switch.

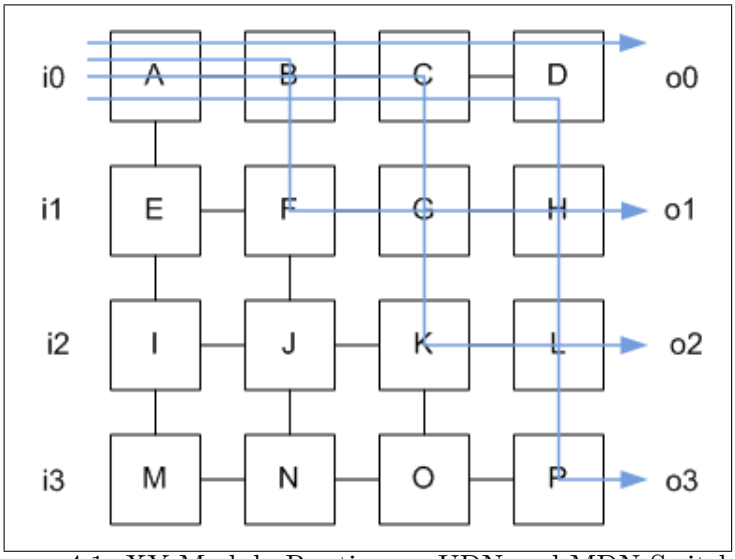


Figure 4.1: XY Modulo Routing on UDN and MDN Switches

Another important parameter of UDN and MDN Routing Algorithms is the T-Value. With the T-Value, the proposed routing algorithms can be made **deterministic**, **oblivious** or even **adaptive**. T-Value is a field in the U(M)DN packet header, and it is used

to alter which column should be in the vertical routing path of a packet. If T-Value is set to 0, the routing is deterministic, meaning that it will always be routed through the same router column. If T-Values are set to a constant value, or random values, the packet could be routed in different columns which make the routing oblivious. If a **load observation unit** is implemented (not a part of this thesis) to compute the T-Values for better load balancing, then the routing would be adaptive.

4.1.1 UDN Routing Algorithm: UDN XY Modulo

UDN XY Modulo Routing Algorithm proposed in [1] makes a balanced distribution of the packets on the columns. There has been made some changes and restrictions on the proposed algorithm in [1], in order to correct a minor mistake and make the algorithm more suitable for hardware implementation. The changes are stressed with the bold font in Listing 4.1. The **conditional equation** of UDN XY Modulo algorithm, **((destAddr mod M) = ((RouterRowAddr + RouterColAddr + T_Value) mod M))**, may evaluate true even if the current packet row is equal to the destination row, which results in an erroneous routing; therefore the conditional equation **if (RouterRowAddr = destAddr)** is added in the code.

The input parameters of the algorithm are **previous router direction (PrevRouter), destination address (destAddr)** and **T-Value**. The internal constants are **Router Row Address, Router Column Address**, and **M**. Modulo M operation requires division in case $M \neq 2^m$ ($m \in \mathbb{N}_0$), and it can be implemented based on the *binary division by a constant integer algorithm*; however in case $M = 2^m$, it can be implemented as a simple bit-selection operation, which avoids the extra cycles and hardware caused by the division operation. This restriction can be extended to allow M = N -1, with some minor modification in the conditional modulo equation, where M is substituted by N, if and only if $N = 2^n$, where $n \in \mathbb{N}_1$.

Listing 4.1: UDN XY Modulo Routing Algorithm

```
case PrevRouter is
when NORTH =>
    if (RouterRowAddr = destAddr) then
        NextRouter      <=  EAST;
    else
        NextRouter      <=  SOUTH;
    end if;
when SOUTH =>
    if (RouterRowAddr = destAddr) then
        NextRouter      <=  EAST;
    else
        NextRouter      <=  NORTH;
    end if;
when WEST =>
    if (RouterRowAddr = destAddr) then      --CORRECTION OVER [1]
        NextRouter      <=  EAST;
    elsif ((destAddr mod M) = ((RouterRowAddr + RouterColAddr + T_Value) mod M)) then
        if (RouterRowAddr > destAddr) then
            NextRouter      <=  NORTH;
```

```
        else
            NextRouter      <=  SOUTH;
        end if;
    else
        NextRouter      <=  EAST;
    end if;
end case;
```

4.1.2 MDN Routing Algorithm: MDN XY Modulo

[1] proposes an MDN Modulo XY routing algorithm that requires the direction of input port whence the packet was injected to the switch and the direction of the destination port whence the packet will be ejected out as input parameters; as a result these data fields have to be included in the U(M)DN packet. We suggest a replacement over the approach in [1]; our suggestion does not require these extra parameters; instead it uses the same input parameters as the UDN XY Modulo, **previous router direction (PrevRouter), destination address (destAddr)** and **T-Value**. The algorithm code is 1/4th of the proposal in [1] in terms of lines of codes, and the hardware is also simpler.

In the UDN switch, the input and output ports are numbered from 1 to N. Since the input ports are only on the West and output ports are only on the East, there is no ambiguity on which side of the switch layout the ports are. However, in the MDN switch, it is needed to compute if a certain I/O port lies on a certain side of the switch, which complicates the routing algorithm. As a solution, we propose that the input/output ports are also indexed with the same scheme as routers in the switch matrix, as shown in Figure 4.2. In this way, we can describe the position of the port on any sides of the switch.

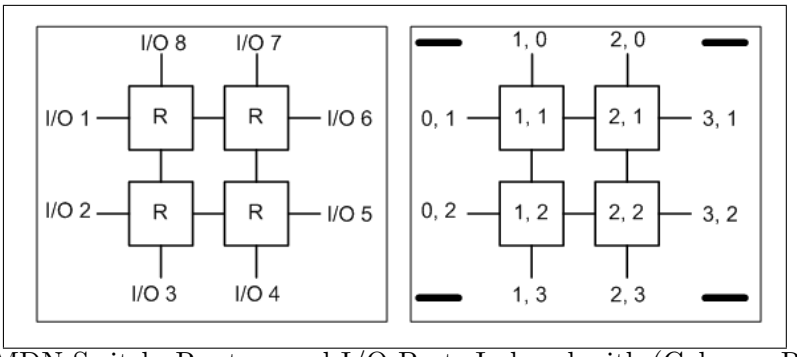


Figure 4.2: MDN Switch, Routers and I/O Ports Indexed with (Column, Row) Indices

In order to translate the output addresses into (column, row) addresses (e.g. 1 → (0, 1)), we need a simple converter (Listing 4.2), where the MdnSwitchM(Direction)S constants refer to the corner point addresses of the switch I/O ports.

Listing 4.2: Switch Output Address to (Row Column) Address Converter

```
if (destAddr < MdnSwitchMSouthS) then
    MdnDestAddr.row     := conv_integer(destAddr)+1;
    MdnDestAddr.col     := 0;
elsif (destAddr < MdnSwitchMEastS) then
    MdnDestAddr.row     := MdnSwitchDim+1;
    MdnDestAddr.col     := conv_integer(destAddr)-MdnSwitchDim+1;
elsif (destAddr < MdnSwitchMNorthS) then
    MdnDestAddr.row     := (3*MdnSwitchDim)-conv_integer(destAddr);
    MdnDestAddr.col     := MdnSwitchDim+1;
else
    MdnDestAddr.row     := 0;
    MdnDestAddr.col     := MdnSwitchM-conv_integer(destAddr);
end if;
```

The MDN XY Modulo routing algorithm we propose (Listing 4.3) is a natural extension of UDN XY Modulo, for output ports placed on multiple sides. XY modulo operation is applied only when a packet is transmitted vertically or horizontally (i.e. from/to switch I/O ports (East→West), (West→East), (North→South), (South→North)). T_Value is very similarly used to vary the columns used in horizontal packet transmission, and the rows used in vertical packet transmission.

Listing 4.3: MDN XY Modulo Routing Algorithm

```
if (destAddrRC.col = RouterColAddr) then
    if (destAddrRC.row < RouterRowAddr) then
        NextRouter  <= NORTH;
    else
        NextRouter  <= SOUTH;
elsif (destAddrRC.row = RouterRowAddr) then
    if (destAddrRC.col < RouterColAddr) then
        NextRouter  <= WEST;
    else
        NextRouter  <= EAST;
else
    case PrevRouter is
        when WEST =>
            if(destAddrRC.col > RouterColAddr) then
                if (destAddrRC.row = (((RouterRowAddr+RouterColAddr+T_Value)
                                        mod MdnSwitchDim) + 1)) then
                    if(destAddrRC.row < RouterRowAddr) then
                        NextRouter  <= NORTH;
                    else
                        NextRouter  <= SOUTH;
                else
                    NextRouter  <= EAST;
            else
                if(destAddrRC.row < RouterRowAddr) then
                    NextRouter  <= NORTH;
                else
                    NextRouter  <= SOUTH;
        when EAST =>
            if(destAddrRC.col < RouterColAddr) then
                if (destAddrRC.row = (((RouterRowAddr+RouterColAddr+T_Value)
                                        mod MdnSwitchDim) + 1)) then
                    if(destAddrRC.row < RouterRowAddr) then
                        NextRouter  <= NORTH;
                    else
                        NextRouter  <= SOUTH;
                else
                    NextRouter  <= WEST;
            else
```

```
                if(destAddrRC.row < RouterRowAddr) then
                    NextRouter  <= NORTH;
                else
                    NextRouter  <= SOUTH;
        when NORTH =>
            if(destAddrRC.row > RouterRowAddr) then
                if (destAddrRC.col = (((RouterRowAddr+RouterColAddr+T_Value)
                                        mod MdnSwitchDim) + 1)) then
                    if(destAddrRC.col < RouterColAddr) then
                        NextRouter  <= WEST;
                    else
                        NextRouter  <= EAST;
                else
                    NextRouter  <= SOUTH;
            else
                if(destAddrRC.col < RouterColAddr) then
                    NextRouter  <= WEST;
                else
                    NextRouter  <= EAST;
        when SOUTH =>
            if(destAddrRC.row < RouterRowAddr) then
                if (destAddrRC.col = (((RouterRowAddr+RouterColAddr+T_Value)
                                        mod MdnSwitchDim) + 1)) then
                    if(destAddrRC.col < RouterColAddr) then
                        NextRouter  <= WEST;
                    else
                        NextRouter  <= EAST;
                else
                    NextRouter  <= NORTH;
            else
                if(destAddrRC.col < RouterColAddr) then
                    NextRouter  <= WEST;
                else
                    NextRouter  <= EAST;
    end case;
```

The algorithm is simulated and verified in MDN switches of various I/O sizes. In the simulations under uniform traffic, the packets are distributed equally to the output ports, which show that the solution proposed in [1] can be securely replaced by this algorithm.

4.2 Scheduling in UDN and MDN Switches

The scheduling unit resolves the contention among the input buffers of a router, competing for the same output port, as well as controlling the arbiter to virtually connect the chosen input buffer to the output port.

We propose the use of some basic scheduling schemes based on Round Robin for UDN scheduling. These are static, dynamic and weighted round robin schemes. In MDN, because of its uniform structure, only static scheduling is implemented.

In this section, we first investigate the traffic load per input/output port pairs in the

UDN and MDN switches, and then explain the proposed scheduling schemes.

4.2.1 UDN Scheduling

The scheduling of the UDN switch is an asymmetric bipartite graph matching problem. This can be formulated as G = (I, O, E) where I denotes input ports (W, N, S), O denotes output ports (E, N, S) and E denotes the edges. The graph is a directed graph. The unmatched graph is presented in Figure 4.3.

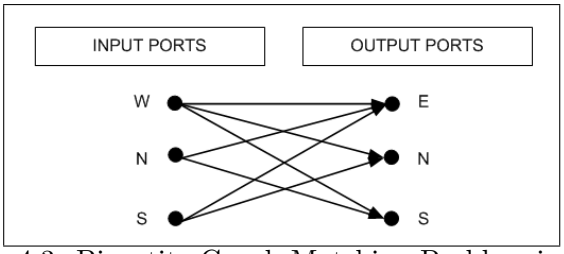


Figure 4.3: Bipartite Graph Matching Problem in UDN

The UDN scheduling algorithms we propose are based on **Round Robin (RR)** scheme. The idea that lies behind RR scheme is that each input buffer in a router is connected to the output port, in fair amounts, such that none of them suffers **starvation** and provide the **optimum throughput**. This can be easily implemented via **circular shift registers**, where the most significant value register is the **RR pointer**, to control the arbiter to connect the corresponding input port to the output port. In the **RR Registers (RRR)**, all of the possible input ports for a given output port are listed (Figure 4.4). When **Shift** signal is set, the registers are shifted according to the **Shift Amount**. There exists an RRR for each Packet Forwarding Unit (PFU). The RRR for the East PFU of a 3 I/O Port UDN Router is illustrated in Figure 4.5. Let us name each period required to successfully transfer a packet from the input buffer of the **current** router to the input buffer of the **next** router a **round**. After each round, RR pointer moves to the next RR value.

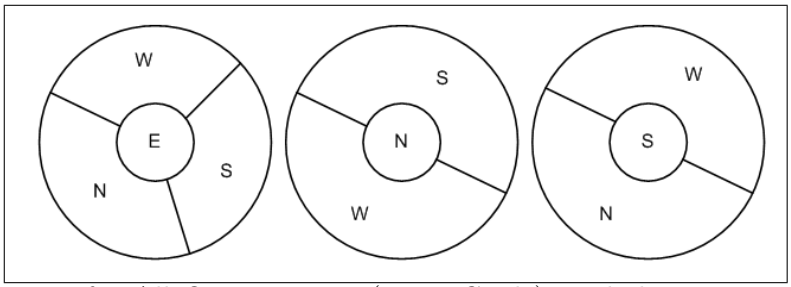


Figure 4.4: RRRs for All Output Ports (Inner Circle), and the Input Ports that can access the respective output ports (Outer Circle)

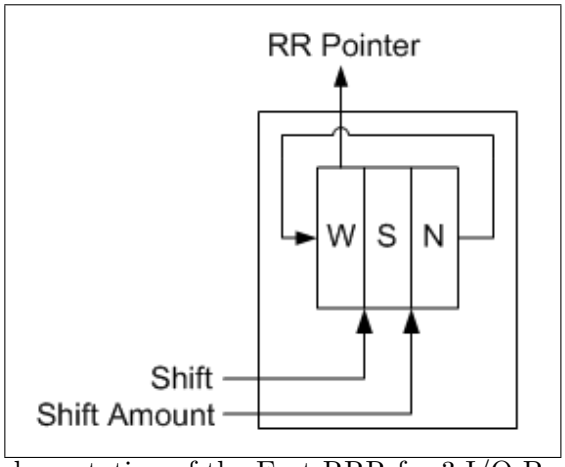


Figure 4.5: Implementation of the East RRR for 3 I/O Port UDN Router

Here is a step-by-step explanation on how the scheduling and arbitration works.

1) The arbiters control the validity and direction of the head-of-line packets in the input buffers they are related to. East Arbiter controls West, North and South buffers; North Arbiter controls West and South buffers; and South Arbiter controls West and North buffers.

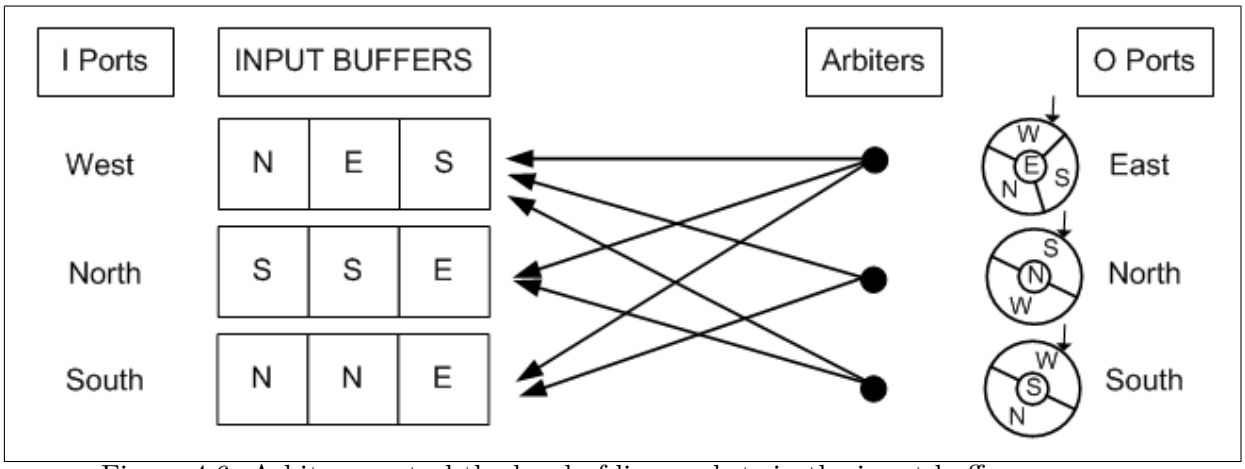


Figure 4.6: Arbiters control the head-of-line packets in the input buffers

2) The matching packet/output port combinations are kept, whereas the others are discarded. At this point, North Arbiter cannot make any connections, South Arbiter can make only one connection to West input buffer, and East Arbiter has to choose in between the South and North input buffers. The decision is made according to the RR pointer (shown with the small arrow on RRRs); the order of priority is (West, North, and South). Therefore, East Arbiter will choose the North input buffer.

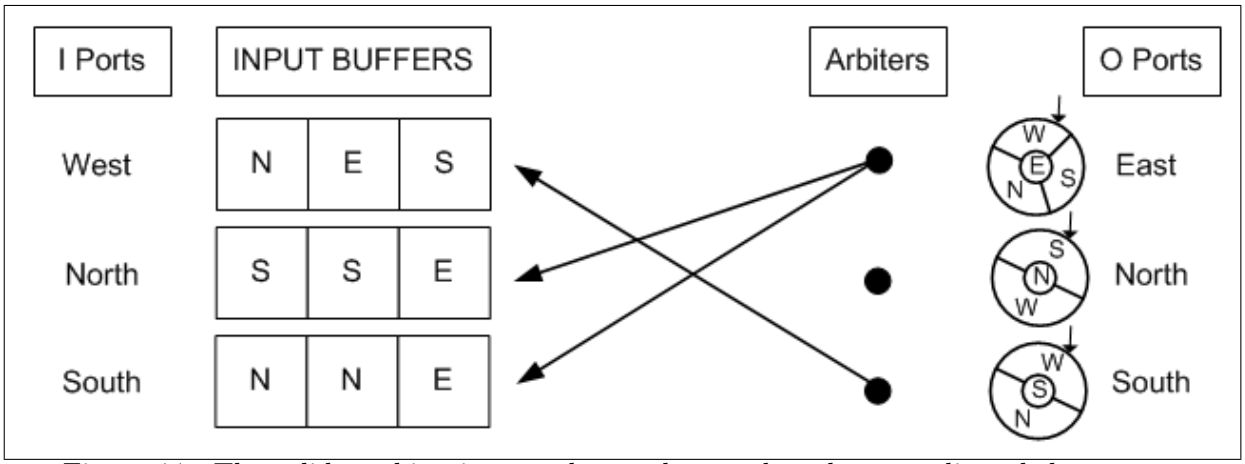


Figure 4.7: The valid combinations are kept, whereas the others are discarded

3) The final connections are made.

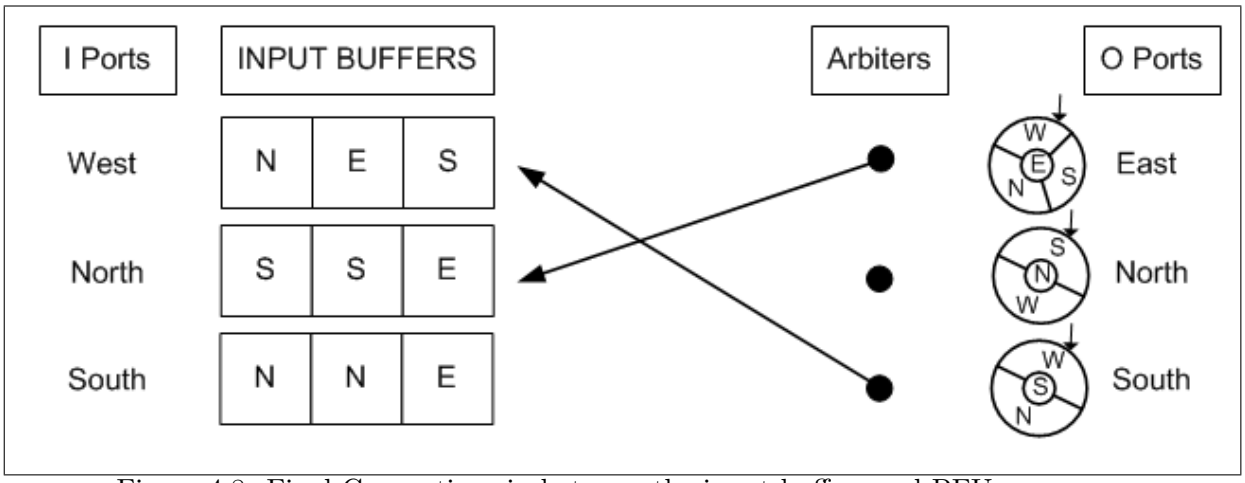


Figure 4.8: Final Connections in between the input buffers and PFUs

In Section 3.5.1, we discussed that design of the arbiter helps reducing the eight types of UDN Routers into two type; one for 2 I/O port and one for 3 I/O port router. Let's consider the case when a certain traffic flow does not exist logically for a certain router type. For example, the North→East flow does not exist in UDN Router Type 2, even though the physical ports exist; therefore this flow is never going to take place. According to our design, regardless of if the East RR pointer points to the North value, the arbiter will check if there is a valid packet at the West, South and North input ports destined to the East output port; once it has the information of valid packets, it will choose them according as in the above example. Since North input buffer cannot have a packet destined to East (in UDN Router Type 2), the next possible input buffers will be granted permission. All of these operations are realized in one cycle, and therefore no extra cycles are spent in case of contention.

We propose three arbitration schemes, based on the RRRs described in the previous paragraphs, to schedule the UDN switch: Static, Dynamic and Weighted Arbitration.

The difference in between the Static and Dynamic arbitration lies in the shift amount, used to shift the RR pointer. In Static Arbitration, the shift amount is always equal to 1; whereas in Dynamic Arbitration, the shift amount depends on the rank of served input port. For example, in the East RRR, the register values are West, North, and South; if the RR pointer points to West, but South (rank = 3) is served (because there are no valid packets in the West input buffer (rank = 1), and nor in the North (rank=2)), the pointer will be shifted by 3. Dynamic arbitration requires a barrel shifter for the hardware implementation, as opposed to the constant shifter in the static arbitration.

The static and dynamic arbitration schemes are far from considering the discussion of traffic flow frequencies in Section 3.5.1. In Table 3.4, it has been shown that router I/O port pairs have different frequencies for transmitting a packet. In order to involve these frequencies into the arbitration scheme, the Round Robin registers need to have weighted values, thus earning its name to the Weighted Arbitration. However, in order to use this arbitration scheme, we have to go one step back, and implement seven types of UDN Routers (one 2 I/O port is enough because of mirrored nature of Type 0 and 7) as opposed to reducing them into 2 types, since the RRR organization is different for each router type.

In Table 3.4, the weights of traffic flows are presented for different router types. In 2 I/O Port Routers (Type 0 and 7), the ratio of West $\rightarrow$ East flow to South(North) $\rightarrow$ East flow is 2x/1x. In 3 I/O Port Routers (Type 1 and 3), the West $\rightarrow$ East flow is also twofold the South(North) $\rightarrow$ East flow. Then, we need to allocate three locations in the East RRR of 2 and 3 I/O port routers, where two of them have the value West, and one South(North).

Please note that the shift amount is always 1 in the weighted arbitration scheme. The ratios and RRR widths change according to the (N, M) values.

4.2.2 MDN Scheduling

The scheduling of the MDN is simpler compared to the UDN switch. First, 4 I/O Port MDN Routers are all the same type, and they allow traffic flow in all input/output port combinations. In addition, the traffic flows have all the same frequencies. Consequently, we only implement Static Round Robin Arbitration for MDN scheduling, which will satisfy the scheduling requirements of the MDN switch due to its uniform router structure and traffic flows.

5 UDN/MDN Synthesis

In this chapter, we present the synthesis results for the UDN and MDN switches. The area and frequency results for specific modules and switches of various (N, M)/(N) sizes are presented. The synthesis is realized on Xilinx ISE 11.1, using Xilinx Synthesis Technology (XST), with the settings "Optimization Goal: Speed", "Optimization Effort: Normal" and "Keep Hierarchy: No". The specific FPGA device is Virtex 5 - XC5VTX240T, FF1759, -2.

The XST reports the area results in terms of **'Number of Slice LUTs'** and **'Number of Slice Registers'**, unlike the results for older platforms like Virtex 4, which reported a single value for slice usage; therefore, all the performance/cost analysis will be reported in 2-tuple, separately for the 'Number of Slice LUTs' and 'Number of Slice Registers'. We do not present the results in terms of logic cell count, gate count or ASIC area size, since the design is made for the reconfigurable platforms, and therefore the conversions of the FPGA metrics using certain formulas are not meaningful, even in terms of providing scientific estimates.

A brief description about the Virtex-5 slice architecture will be that it consists of 4 6-bit input look-up tables (LUTs) and 4 Flip-Flops (FFs). A 6-bit input LUT can implement a more complex function compared to a 4-bit input LUT (as in older FPGA platforms), increasing the frequency of the circuitry and decreasing the resource usage at the same time.

A question might arise why we present synthesis results, which are only an estimate of the "place & route" results. There are three main reasons: First, our switch interface involves very wide ATM ports which cannot be mapped onto an FPGA without serializer and deserializer units; however, the presence of these units would affect the place & routing results. Second, the whole synthesis, mapping and place & routing process takes very long times for large switch sizes; because we want to obtain results over a wide range of (N, M)/(N) values, running all processes makes it very impractical for the purposes of this thesis. Third, XST tool, starting from version 9.1, claims that the correlation in between the synthesis and place & route tools have been improved, resulting in synthesis results with higher quality.

Please note that, in the designs, we did not use any embedded FPGA resources like operational modules or block rams; the modules are described in a way that they would only infer Slice LUTs and Slice Registers, but not other embedded FPGA resources. In this way, a better performance/cost analysis can be carried out.

5.1 UDN Synthesis Results

The modules that constitute the UDN switch have different tasks and therefore different weights on the combinational and sequential circuits. This can be observed by comparing the weights of the number of LUTs and Registers for any module. The synthesis results of the individual modules are given in Table 5.1. The input buffer depth is set to 2.

Table 5.1: Synthesis Results for individual UDN Modules

	Size (Number of Slice ...)		Maximum Frequency (MHz)
Module Names	**... LUTs**	**... Regs**	
Input NI Buffer	1278	853	734.700
Input NI	359	857	597.229
Output NI	450	857	510.843
XY MODULO, UDN	2	-	1426.53
Router Buffer	531	1016	508.414
Router (2 I/O Port)	1679	2045	269.485
Router (3 I/O Port)	3073	3073	257.107

From the Table 5.1.1, the following information can be obtained:

- The critical path of the switch is on the router with highest port number (i.e. 3 I/O, in UDN)
- Considering that there are 2 input buffers in the 2 I/O port Router, the remaining LUTs are used for the implementation of the arbiter, routing logic and flow control logic. (≈600 LUTs for 2 I/O Port Router)
- Almost all registers are used in the input buffers for the NIs and Routers, except some simple counters used in the flow control. There are also no pipe-stages other than the buffers themselves.

The synthesis for the UDN switch is carried out for a wide range of (N, M) combinations, in order to make the area (cost) and frequency (performance) exploration over a large number of samples. The resulting frequencies, Number of Slice LUTs and Number of Slice Registers are given in Table 5.2, Table 5.3, and Table 5.4. In the tables, the blue colors (light gray in B&W copy) imply that the switch fits on the FPGA chip it was synthesized for; whereas the red colors (dark gray in B&W copy) imply that the switch uses more than 100% resources. In order to obtain these results, the constraint on 100% was removed. These circuits can be hypothetically mapped on an FPGA with greater slice count. The cells with no values are invalid (N, M) combinations. The respective charts for the tables are also presented.

The **critical path** in the switch, which determines the operational frequency of the all circuitry, is on the 3 I/O Port Router, and it starts from the Round Robin registers of

any arbiter, passes through the flow control logic, then through the Read Enable signal that decrements the Read Pointer in the Input Buffer and ends on the Status Counter, which determines if the buffer is FULL, EMPTY or in between.

Table 5.2: Frequency Results for various (N, M) Switches in MHz

N\M	1	2	3	4	7	8	15	16	31	32
2	313.593	313.593								
3	292.583	279.698								
4	292.583	292.583	287.861	287.979						
5	285.767	285.486		285.382						
8	285.312	285.034		281.163	281.163	281.163				
12	256.652	256.567		257.138		257.138				
16	247.419	241.452		244.568		244.121	240.785	235.468		
32	220.146	215.685		213.126		211.589		211.589	209.466	207.654

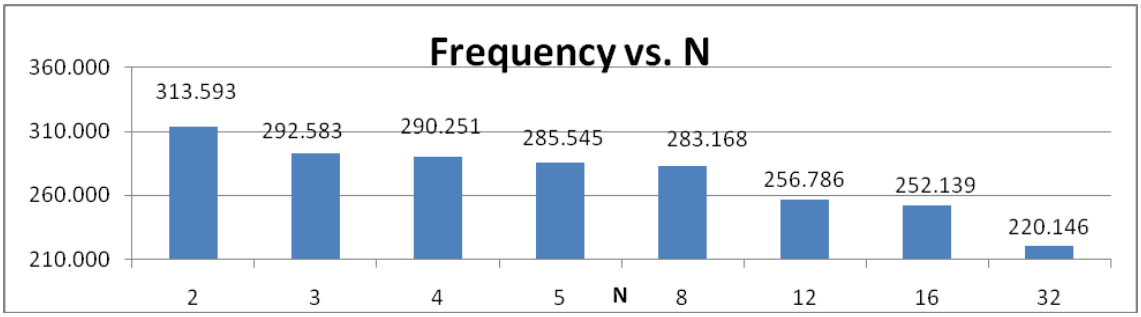


Figure 5.1: Frequency Results for various (N, 1) Switches

The frequency depends on the number of switch I/O Ports (N value), but not the M value. There are very small variances among the values for different M, as a result of the heuristic algorithms used in the synthesis. Even though the critical path does not change from N value to N value, the routing delay increases since the usage of FPGA resources also increase, decreasing the operational frequency of the circuit. In an **ASIC synthesis**, there would be very little difference among different N values as well, since the critical path remains the same.

The number of slice LUTs is the parameter that defines the area/cost of the combinational logic, in FPGA terminology. In Table 5.3, the synthesis results for this metric are listed. Two graphs are presented, one plotted versus M and the other versus N.

In Figure 5.2, it can be observed that the increase in the charts are linear, except when N=M, where there is very small increase. An important note about the synthesis is that the T-Values were tied to 0 during the process; as a result, when N=M, the final column does not infer full arbiters, but simplified modules which support only West'East

Table 5.3: Number of Slice LUTs for various (N, M) Switches

N\M	1	2	3	4	7	8	15	16	31	32
2	5148	6376								
3	9802	16796								
4	14415	24987	31759	37106						
5	18998	33783		62375						
8	32757	59140		109007	179978	184203				
12	51404	83915		157282		330422				
16	69883	116600		212202		412382	762421	815425		
32	127953	227833		428484		829785		1632388	3137269	3237595

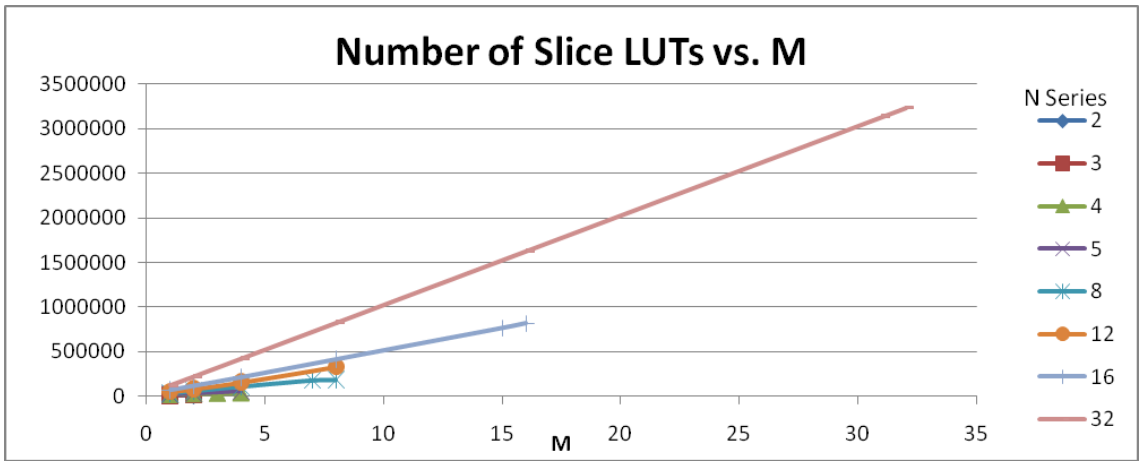


Figure 5.2: Number of Slice LUTs vs. M

traffic flow. This is not something we have done intentionally; the synthesis optimization algorithms automatically simplify the arbitration modules, due to the reasoning explained on Figure 3.14. The increase is also linear for the charts in Figure 5.3. This implies that as N x M product is increased, the cost increases quadratically, resulting in high resource costs.

In Figure 5.4 and Figure 5.5 it is shown that the "Number of Slice Registers" increases linearly for plots vs. M and N, similar to the "Number of Slice LUTs" charts.

5.2 MDN Synthesis Results

The MDN switch inherits many modules from the UDN switch. MDN XY Modulo Unit, Virtual Channel and 4 I/O Port Router are the main additions to the building blocks. The synthesis results of the individual modules are given in Table 5.5.

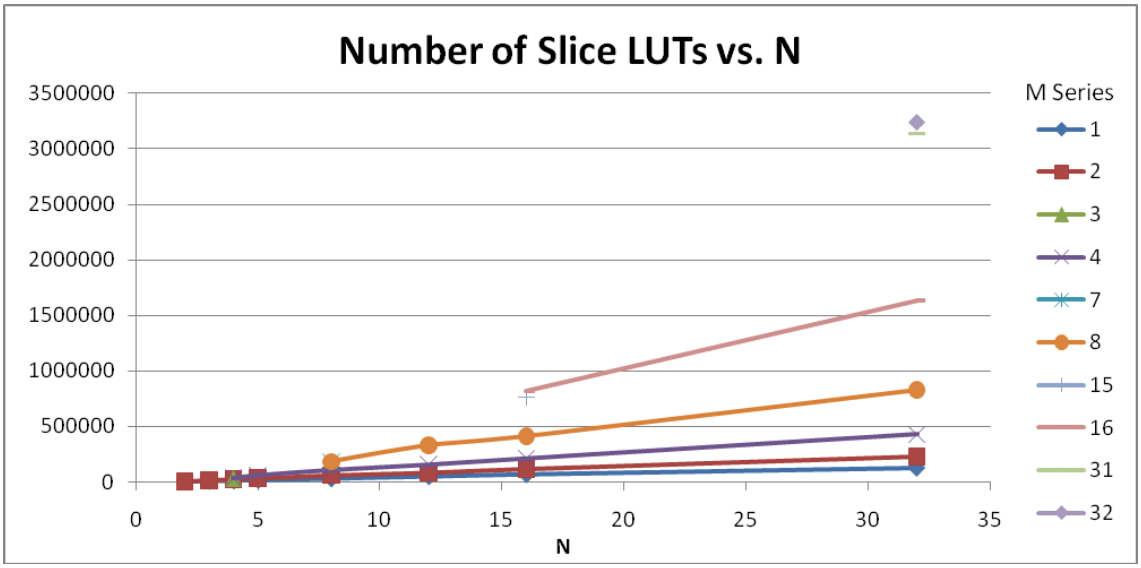


Figure 5.3: Number of Slice LUTs vs. N

Table 5.4: Number of Slice REGs for various (N, M) Switches

N/M	1	2	3	4	7	8	15	16	31	32
2	7516	9562								
3	12301	19467								
4	17089	27324	37590	41655						
5	21875	35184		61797						
8	36236	58771		103838	171508	179707				
12	55396	90251		159977		299236				
16	74545	121683		215954		404332	734114	781216		
32	151129	247409		439970		825090		1595332	3039532	3135810

Table 5.5: Synthesis Results for individual MDN Modules

	Size (Number of Slice ...)		Maximum Frequency (MHz)
Module Names	**... LUTs**	**... Regs**	
NI I Buffer	1278	853	734.700
Input NI	359	857	597.229
Output NI	450	857	510.843
XY MODULO, MDN	2	-	-
Router I Buffer	531	1016	508.414
Virtual Channel	532	1016	508.414
Router (4 I/O Port)	6197	4112	255.115

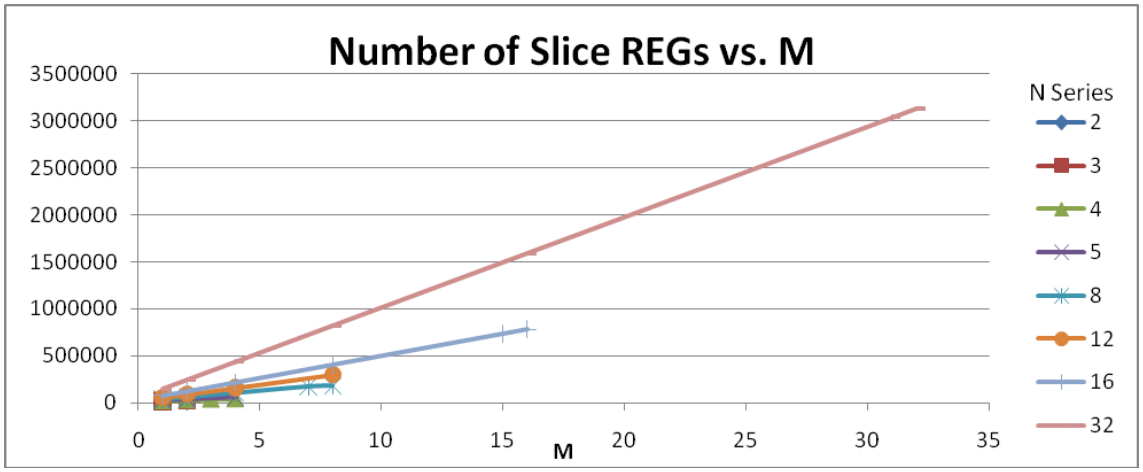


Figure 5.4: Number of Slice REGs vs. M

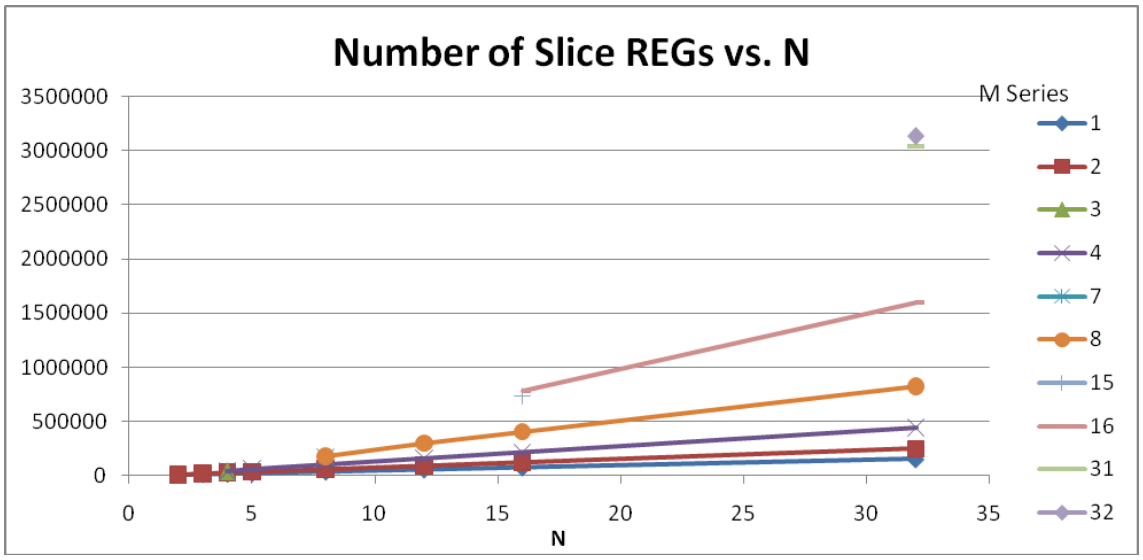


Figure 5.5: Number of Slice REGs vs. N

The synthesis for the MDN switch is carried out for a wide range of (N) values, in order to make the area (cost) and frequency (performance) exploration over a large number of samples. The resulting frequencies, Number of Slice LUTs and Number of Slice Registers are given in Table 5.6, Table 5.7, and Table 5.8.

The critical path in the MDN switch, which determines the operational frequency of the all circuitry, is on the 4 I/O Port Router, and it passes through the same exact signals as in 3 I/O Port UDN Router.

Table 5.6: Frequency Results for various MDN Switches

N	Frequency (MHz)
4	250.591
8	240.667
12	233.598
16	212.227
20	201.000
24	191.150
28	185.564
32	173.214

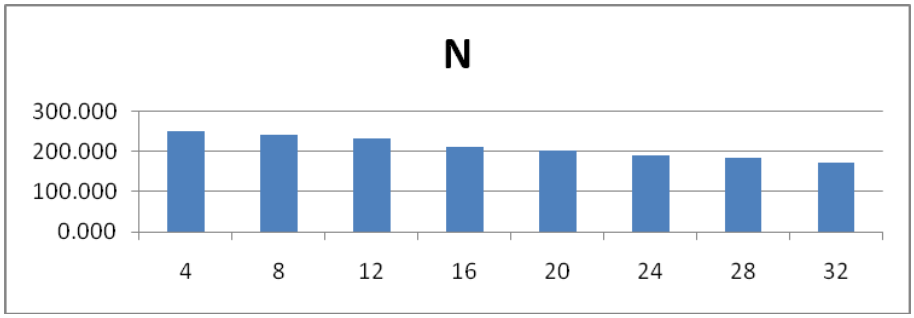


Figure 5.6: Frequency vs. N, in MDN

Table 5.7: Number of Slice LUTs for various MDN Switches

N	Number of Slice LUTs
4	8933
8	30387
12	62678
16	108996
20	166373
24	229778
28	312343
32	410697

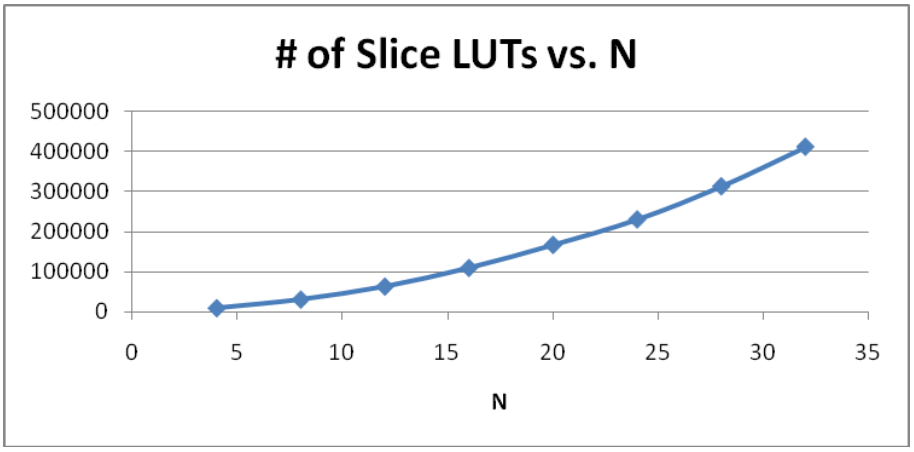


Figure 5.7: Number of Slice LUTs vs. N, in MDN

Table 5.8: Number of Slice Registers for various MDN Switches

N	Number of Slice Regs
4	10969
8	30153
12	57630
16	93203
20	137080
24	189168
28	249480
32	318016

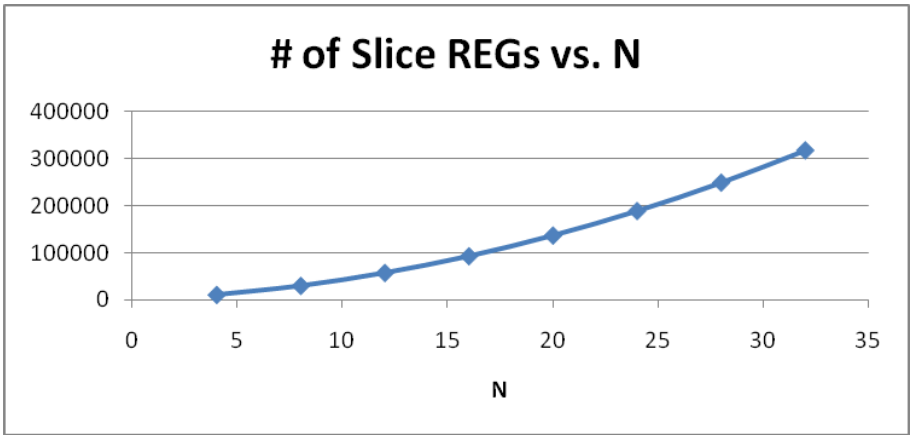


Figure 5.8: Number of Slice Registers vs. N, in MDN

5.3 UDN/MDN Comparison

The UDN vs. MDN switches comparison is tabulated in Table 5.9. Even though the 4 I/O Port MDN Router consumes twice the amount of resources as the 3 I/O Port UDN Router, and 3.5 times as the 2 I/O Port UDN Router, the comparison of the overall UDN and MDN switches show that MDN is much more cost efficient for a given N value (number of switch input/output ports). On the other hand, the operational frequencies of the MDN switches of various sizes are below the operational frequencies of the UDN switches, which affects the performance. The costs of the UDN switch increase quadratically, as the N × M product is increased, unlike the MDN switch which increases subquadratically ($(N/4)^2$). Therefore, in terms of costs, MDN is preferable over UDN.

The performance does not only depend on the frequency, but throughput as well, which will be explored in Chapter 6 and 7.

Table 5.9: UDN and MDN Switch Comparison

	Max Number of Router Ports	Max Router Size (Number of S. LUTs/REGs)	Number of Routers	Cost Increase w.r.t. Number of Routers	Max Frequency (MHz, N=4)
UDN	3	3073/3073	N^2 - N (i.e. M=N-1)	Quadratic	290
MDN	4	6197/4112	$\left(\frac{N}{4}\right)^2$	Subquadratic	250

6 UDN/MDN Simulations

In this chapter, we present the simulation results of various sizes of UDN and MDN switches, the arbitration schemes we proposed, and the usage of T-Value for load balancing.

The simulations in this chapter are all carried out on fully synthesizable circuitry, with the exception of packet generation units; therefore the simulations present reliable information on the performance, which will be used together with the frequency results in Chapter 5, to yield performance metrics of the switches.

We have used Modelsim, as the HDL simulation tool.

6.1 Simulation Environment

We have implemented two modules as a part of the testbench; namely, ATMgenerator and ATMsink. ATMgenerator uses open-source SynthWorks' Random Library (VHDL) to generate uniform and weighted pseudo-random values, which are used as the destination address for the ATM packets. ATMgenerator module outputs the packet information on a text file, for each generated ATM packet. This information is composed of 1) Destination Address, 2) Source Address, and 3) Packet ID. This information is also added to the ATM packet's payload, for verification purposes; ATMsink outputs the ejected packet information on a text file, and the two files are compared such that it can be verified that the injected and ejected ATM packets correspond to each other. In this way, we also obtain the number of injected and ejected packets, at any given time in the simulation, in order to compute the throughput performance of the system.

6.1.1 Traffic Types

The simulation is carried out for both of the UDN and MDN switches, under the **Bernoulli Uniform** and **Unbalanced (Weighted) Traffic.**

The **Bernoulli Uniform Traffic** is commonly used for testing of the network switches. In the Bernoulli Uniform Traffic, the probability that an input packet is destined to a certain output port is equal to the probability of any other output ports. This ensures that the traffic load is distributed equally among all input/output port pairs.

The **Unbalanced (Weighted) Traffic** requires that the traffic is not distributed

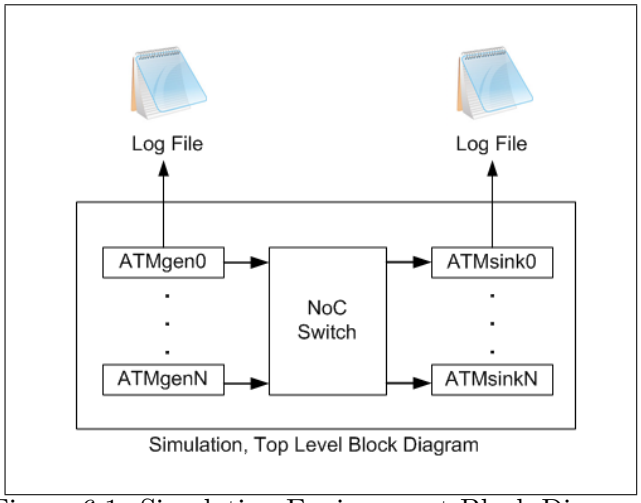


Figure 6.1: Simulation Environment Block Diagram

equally on the output ports as in Bernoulli Uniform Traffic, but it should be distributed with a certain weight on some of the output ports. In this thesis we define the weight in the following scheme: For each generated packet, the probability that the packet is destined to the output port Oi is equal to (i / ($\sum_{n=1}^{N} n$); for example if the switch I/O port number, N = 4, then a packet will have the probability of 1/10 for O1, 2/10 for O2, 3/10 for O3 and 4/10 for O4.

6.2 UDN Simulations

The UDN simulations are carried out for all (N, M) switch size combinations, for which the synthesis results are reported in Section 5.1. In this way, the corresponding "Number of Average Packets / Cycle" is calculated for each (N, M) combination.

The simulations are carried out for 25000 cycles (250 s, for 100 MHz), with input buffer width of two and static arbitration. Each 100 cycles, the simulator computes the average number of packets injected to the switch per input port, the average number of packets ejected from the switch per output port, "average number of packets per output port per cycle", and "percentage of sent packets" (number of output packets / number of input packets × 100). We cannot present this data in a table due to its large size; however the charts from Figure 6.2 to Figure 6.5 present the respective charts for the tables corresponding to the simulations of UDN switches (2,1) and (3,1). As it can be observed on the charts, the values increase (or/and decrease) until they saturate to a final value. The region before the saturation is the "cold start" region, during when the switch routers' buffers are filled, as the packets proceed through the switch; the saturated values represent the switches under full performance. "Percentage of sent packets" charts show how the number of input packets to output packets ratio also saturates after the cold start period; the final saturated value indicates the latency of the switch.

In the following subsection, we will only provide the saturated values, instead of presenting the tables or the charts for the simulations. The saturated values are calculated as the average of the values in between 20'000th and 25'000th cycles.

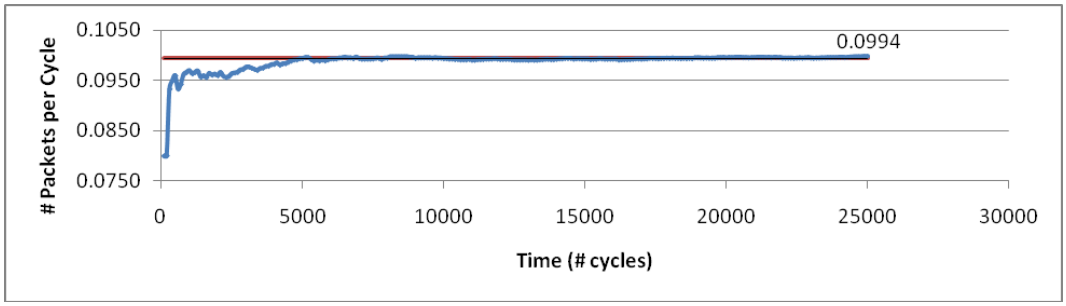


Figure 6.2: Average Number of Packets/Output Port/Cycle, UDN (2,1)

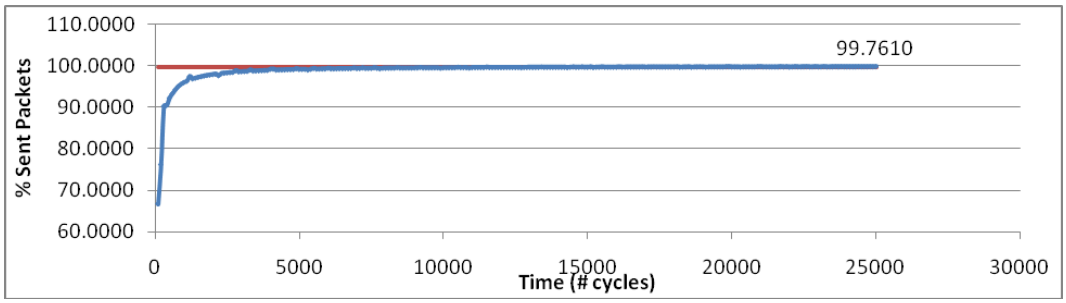


Figure 6.3: % Sent Packets, UDN (2,1)

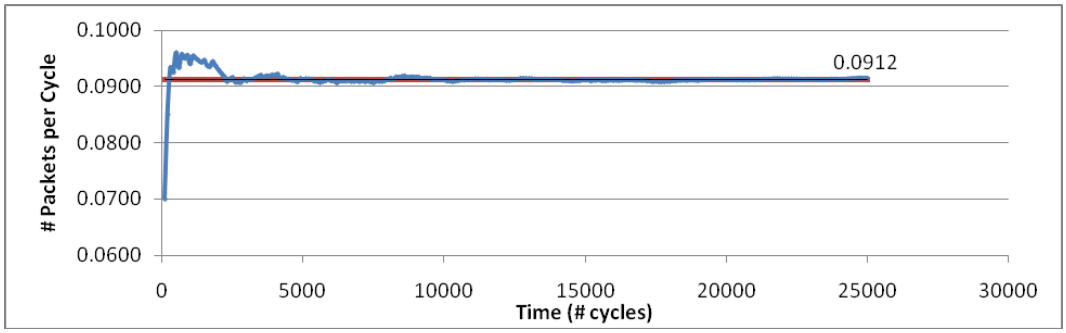


Figure 6.4: Average Number of Packets/Output Port/Cycle, UDN (3,1)

6.2.1 Simulations under Bernoulli Uniform Traffic

We provide the simulation results of UDN switches with possible (N, M) values, under Bernoulli Uniform Traffic, in Table 6.1 - Table 6.2, and their respective graphs in Figure 6.6 - Figure 6.7.

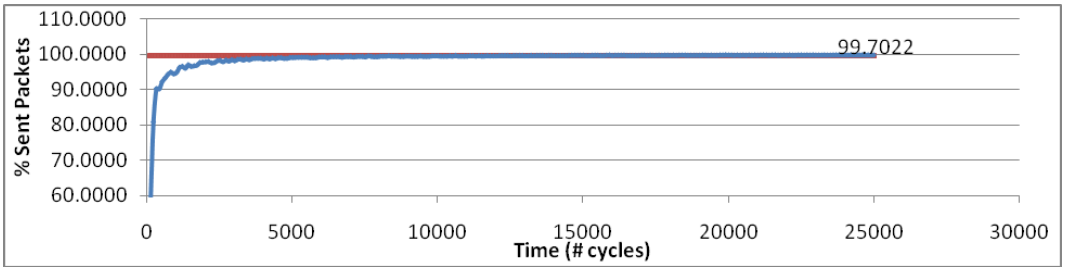


Figure 6.5: % Sent Packets, UDN (2,1)

Table 6.1: Average Number of Packets/Output Port/Cycle, after saturation

N\M	1	2	3	4	7	8	15	16	31	32
2	0.09942	0.09937								
3	0.09118	0.09293								
4	0.08272	0.08378	0.08829	0.08824						
5	0.07182	0.08318		0.08720						
8	0.04609	0.06507		0.08103	0.08731	0.08726				
12	0.02931	0.04746		0.07247		0.08447				
16	0.02062	0.03559		0.06064		0.08315	0.08977	0.08973		
32	0.00920	0.01561		0.03054		0.06502		0.08721	0.09343	0.09338

Table 6.2: % Sent Packets, after saturation

N\M	1	2	3	4	7	8	15	16	31	32
2	99.7610	99.7115								
3	99.7022	99.6551								
4	99.6498	99.5827	99.5410	99.4901						
5	99.5741	99.5570		99.4547						
8	99.2607	99.3627		99.3406	99.1847	99.1344				
12	98.7694	99.0699		99.1846		99.0383				
16	98.2020	98.7412		98.9474		98.9245	98.5530	98.5075		
32	95.9586	97.2333		98.1099		98.3956		98.1606	97.3444	97.2954

"Number of Average Packets/Output Port/Cycle" is an important factor of the "throughput performance" function of the switch, together with the "operational frequency". As M is increased for constant N series, the "Number of Average Packets/Output Port/Cycle" value increases logarithmically, until it saturates at M = N - 1. When M = N, the value decreases slightly, because of the delay caused by the abundant router column.

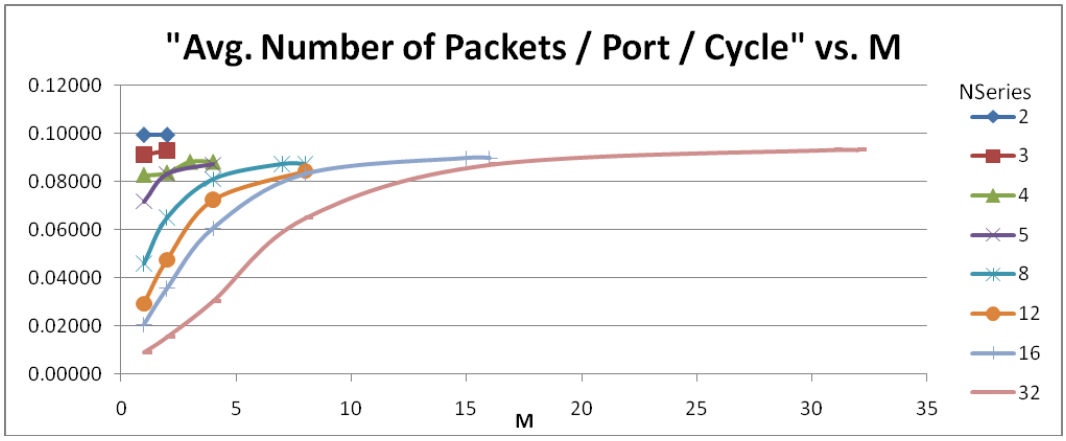


Figure 6.6: Average Number of Packets/Output Port/Cycle vs. M, after saturation

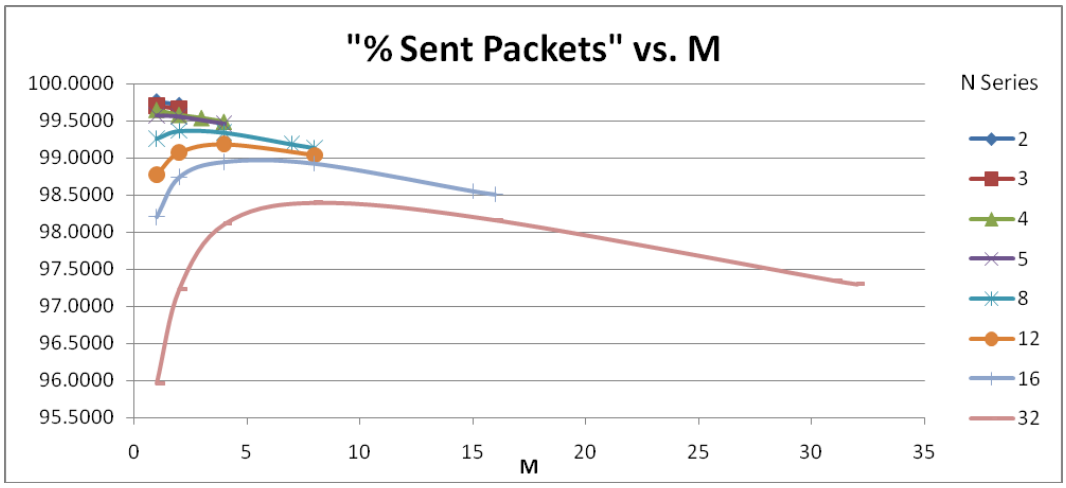


Figure 6.7: % Sent Packets vs. M, after saturation

The behavior of the series in "% Sent Packets" vs. M graph can be explained with two different phenomena. It is observed that the series first increase logarithmically and then decrease linearly; the increase is caused by the fact that when M is too small, there is **congestion** over the routers to route the packets from/to relatively large number of I/O ports of the switch; as M increases the congestion decreases, and thus the series increase. The reason for the series to decrease after a certain M values is that the number of columns, and therefore the **number of hops** for the packet to travel, suppresses the first phenomena; as M increases, there are more stages (routers) that the packets must be routed through, and therefore the series start to decrease.

In Figure 6.6, we have shown that the constant N series logarithmically increase, when M is also increased. In order to look to the same data set from another perspective, the "Number of Average Packets/Output Port/Cycle" vs. N graph with constant M series is plotted (Figure 6.8); in this new graph, the constant M series decrease as N is

increased. This means that as N is increased, the switch's **performance per output port** decreases.

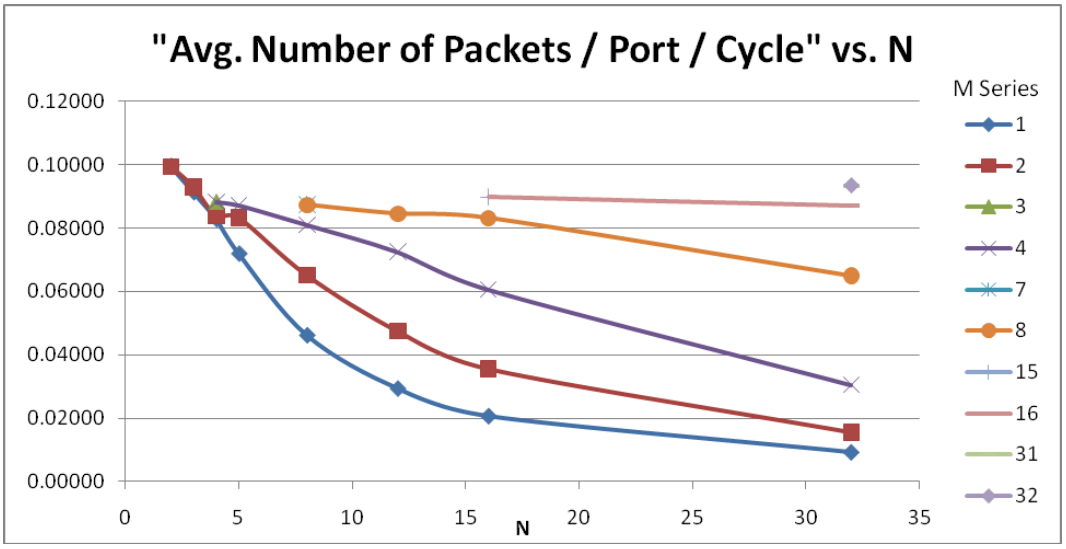


Figure 6.8: Average Number of Packets/Output Port/Cycle vs. N, after saturation

In order to evaluate the real performance of a switch, "Number of Average Packets/Output Port/Cycle" parameter should be transformed into "Number of Packets/Cycle" (Table 6.3) by multiplying the values with the corresponding N value, such that the overall performance of the switch is obtained. This new parameter is plotted vs. M (Figure 6.9) and N (Figure 6.10). N series increase logarithmically until M = N - 1, and then slightly decrease at M = N. M series increase slightly for small M values, and linearly for greater M values; when M is too small compared to N, the traffic on the switch gets congested, and therefore there is little performance increase, and even some decrease for greater N values. The graphs show that the UDN switch scales well under the Bernoulli Uniform Traffic.

Table 6.3: Number of Packets/Cycle

N\M	1	2	3	4	7	8	15	16	31	32
2	0.198848	0.198749								
3	0.273554	0.278795								
4	0.330874	0.335135	0.353146	0.352966						
5	0.359095	0.415884		0.435995						
8	0.368747	0.520553		0.648276	0.698460	0.698106				
12	0.351736	0.569522		0.869665		1.013682				
16	0.329864	0.569373		0.970280		1.330376	1.436374	1.435711		
32	0.294555	0.499536		0.977318		2.080759		2.790609	2.989712	2.988207

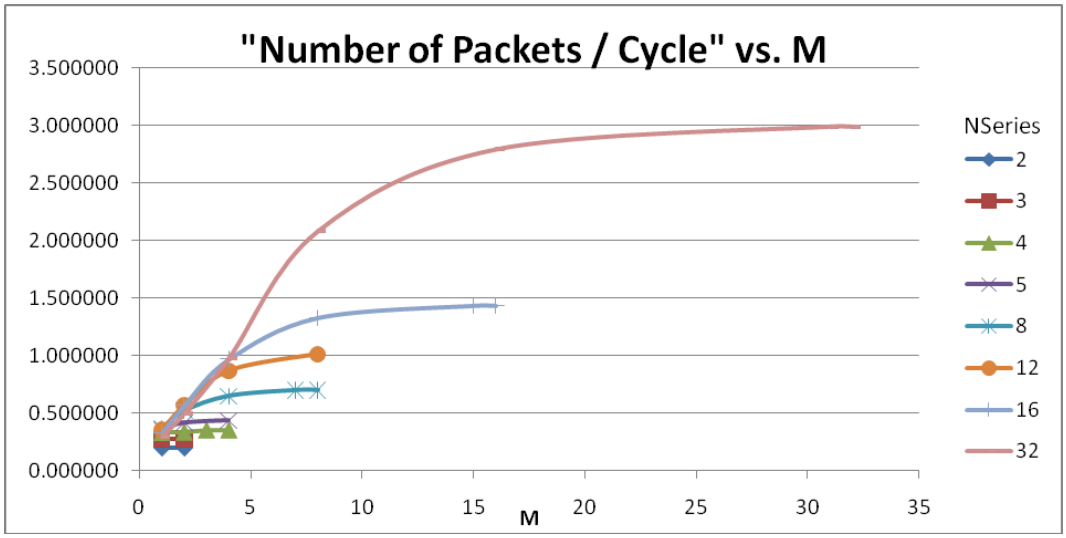


Figure 6.9: Number of Packets/Cycle vs. M, after saturation

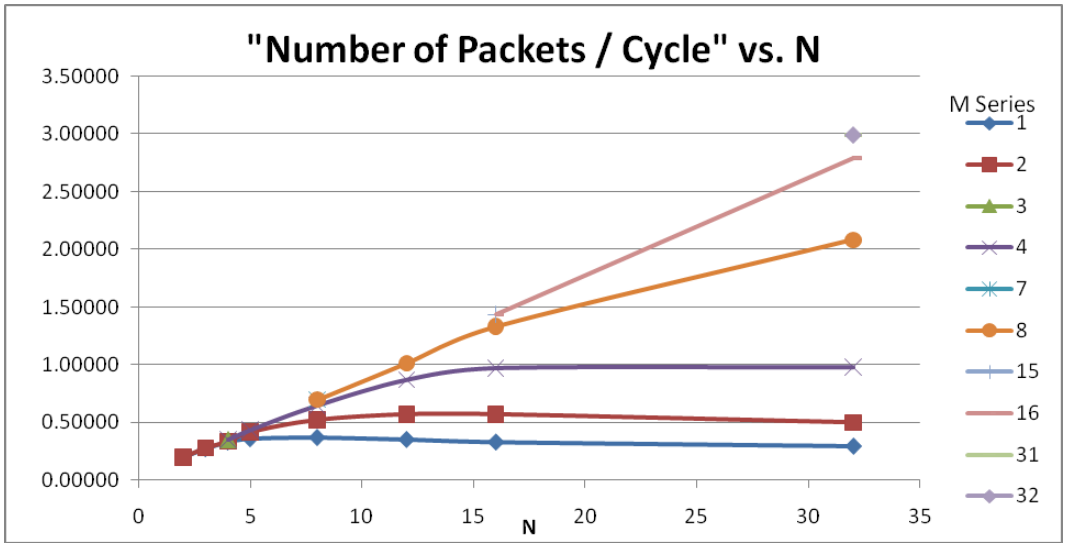


Figure 6.10: Number of Packets/Cycle vs. N, after saturation

6.2.2 Simulations under Unbalanced (Weighted) Traffic

We provide the simulation results of UDN switches with possible (N, M) values, under Unbalanced (Weighted) Traffic, in Table 6.4 - Table 6.5, and their respective graphs in Figure 6.11 - Figure 6.12.

The behavior of the charts in Figure 6.11 - Figure 6.12 are already explained in the previous subsection. Here, we will only discuss the differences from the results under Bernoulli Uniform Traffic.

Table 6.4: Average Number of Packets/Output Port/Cycle, after saturation

N\M	1	2	3	4	7	8	15	16	31	32
2	0.087803	0.087770								
3	0.074663	0.077762								
4	0.066354	0.065770	0.070912	0.070885						
5	0.058845	0.062711		0.068980						
8	0.042246	0.049831		0.056598	0.063482	0.063458				
12	0.029014	0.039382		0.048395		0.053644				
16	0.020989	0.032854		0.042931		0.050547	0.057758	0.057735		
32	0.009449	0.016877		0.027694		0.036534		0.044222	0.052102	0.052084

Table 6.5: % Sent Packets, after saturation

N\M	1	2	3	4	7	8	15	16	31	32
2	99.783	99.746								
3	99.728	99.676								
4	99.676	99.603	99.552	99.515						
5	99.611	99.546		99.458						
8	99.429	99.417		99.323	99.119	99.082				
12	99.116	99.223		99.192		98.982				
16	98.733	99.052		99.063		98.829	98.296	98.257		
32	97.128	98.177		98.603		98.426		97.903	96.568	96.536

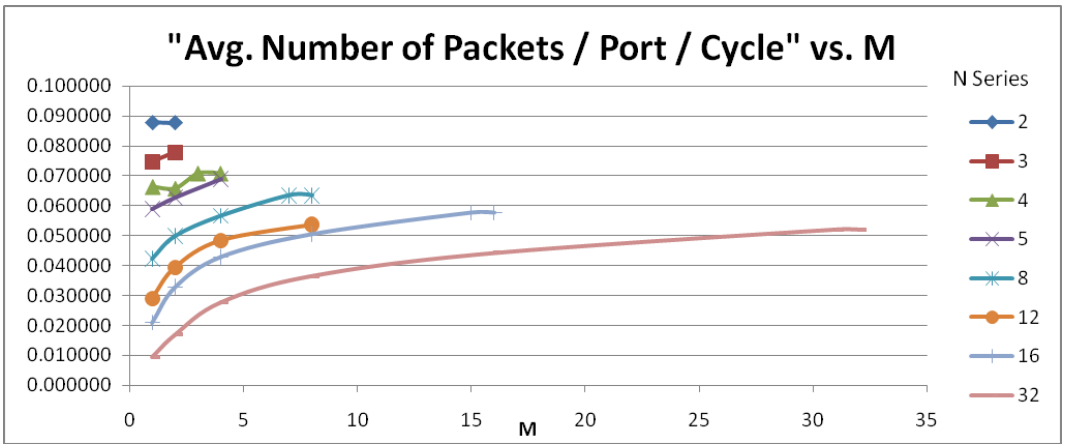


Figure 6.11: Average Number of Packets/Output Port/Cycle vs. M, after saturation

Figure 6.6, the performance of greater N series gets very close to the performance of smaller N series, as M increases and the series saturate. However, in Figure 6.11, greater

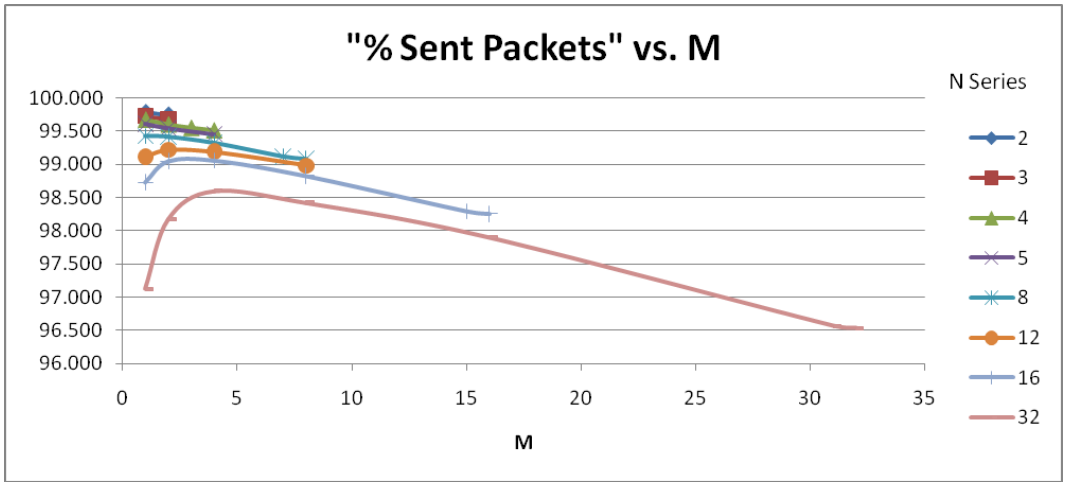


Figure 6.12: % Sent Packets vs. M, after saturation

N series perform worse than the smaller N series. For example, "Average Number of Packets/Output Port/Cycle" is equal to 0.0994 for Switch (2, 1) and 0.0934 for Switch (32, 32) under Bernoulli Uniform Traffic, where the difference is much smaller, as opposed to the difference in between 0.0878 for Switch (2, 1) and 0.0521 for Switch (32, 32) under Unbalanced (Weighted) Traffic. In a similar way, the values of N series are smaller when M = N than when M = 1, under Unbalanced (Weighted) Traffic, as opposed to the simulation under Bernoulli Uniform Traffic, where the values when M=N are greater than when M=1.

In Figure 6.13, "Average Number of Packets/Output Port/Cycle" vs. N graph is presented.

Following the discussion in the previous subsection, we compute "Number of Packets/Cycle" (Table 6.6), in order to obtain the overall performance of the switches. The charts "Number of Packets/Cycle" vs. M and vs. N are presented in Figure 6.14) and N (Figure 6.15. The graphs show similar behavior to the simulations under Bernoulli Uniform Traffic, and therefore indicate that UDN Switch scales well under the Unbalanced Traffic.

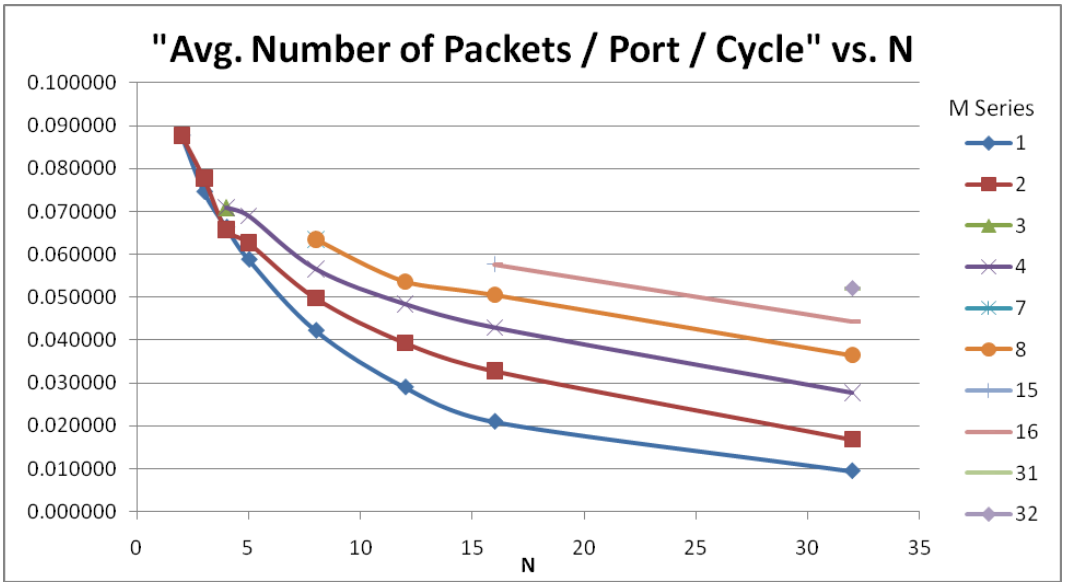


Figure 6.13: Average Number of Packets/Output Port/Cycle vs. N, after saturation

Table 6.6: Number of Packets/Cycle

N\M	1	2	3	4	7	8	15	16	31	32
2	0.198848	0.198749								
3	0.273554	0.278795								
4	0.330874	0.335135	0.353146	0.352966						
5	0.359095	0.415884		0.435995						
8	0.368747	0.520553		0.648276	0.698460	0.698106				
12	0.351736	0.569522		0.869665		1.013682				
16	0.329864	0.569373		0.970280		1.330376	1.436374	1.435711		
32	0.294555	0.499536		0.977318		2.080759		2.790609	2.989712	2.988207

6.3 MDN Simulations

The MDN Switch simulations are carried out for all (N), for which the synthesis results are reported in Section 5.2. In this way, the corresponding "Average Number of Packets / Cycle" is calculated for each N value. The simulations are performed in the same way as UDN simulations, under the Bernoulli Uniform Traffic and Unbalanced (Weighted) Traffic, and the resulting charts are presented in the following subsections.

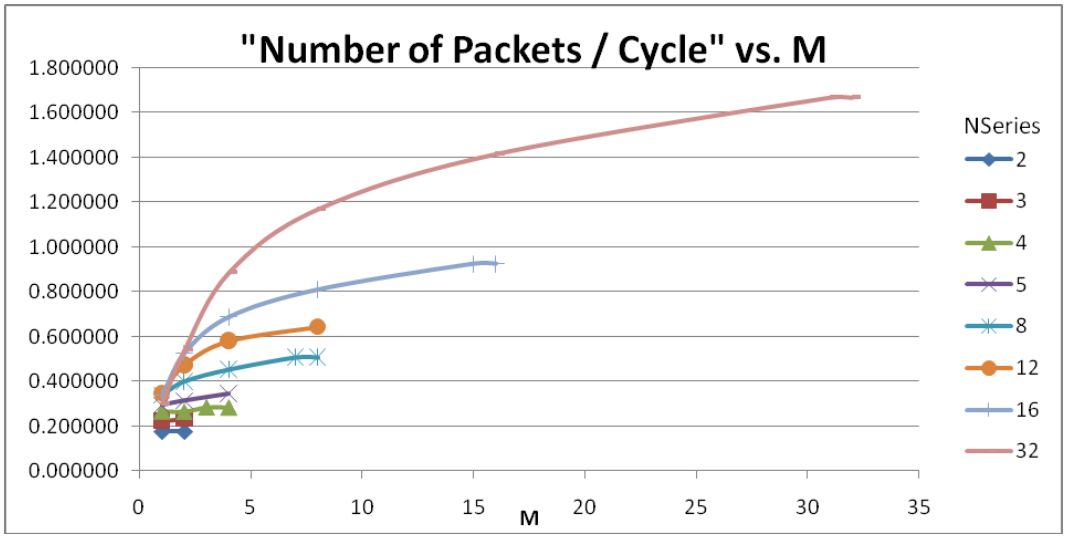


Figure 6.14: Number of Packets/Cycle vs. M, after saturation

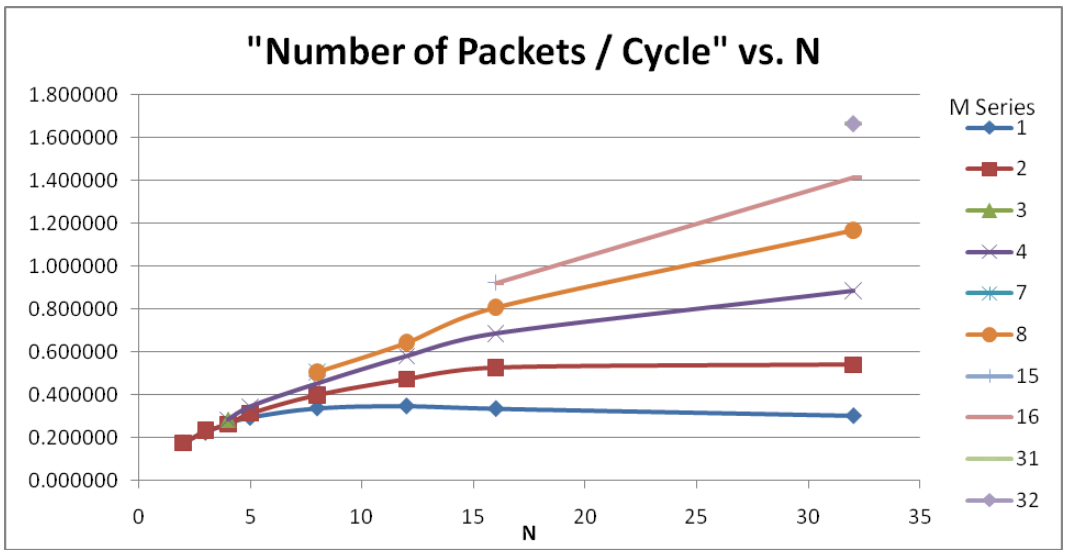


Figure 6.15: Number of Packets/Cycle vs. N, after saturation

6.3.1 Simulations under Bernoulli Uniform Traffic

The simulation results of MDN Switches with possible (N) values, under Bernoulli Uniform Traffic, are presented in Table 6.7 and its respective graph in Figure 6.16.

In Figure 6.16, as the N values increase, there is first some decrease in the values of "Average Number of Packets/Output Port/Cycle" until N=12, and then some logarithmic increase. The decrease is because of the multi-hop nature of the MDN switch: when N=4, there is one single router, and therefore no congestion in the traffic flows;

Table 6.7: Average Number of Packets/Output Port/Cycle, after saturation

N	# Avg. Packets / Port / Cycle
4	0.0813
8	0.0756
12	0.0566
16	0.0579
20	0.0585
24	0.0596
28	0.0602
32	0.0602

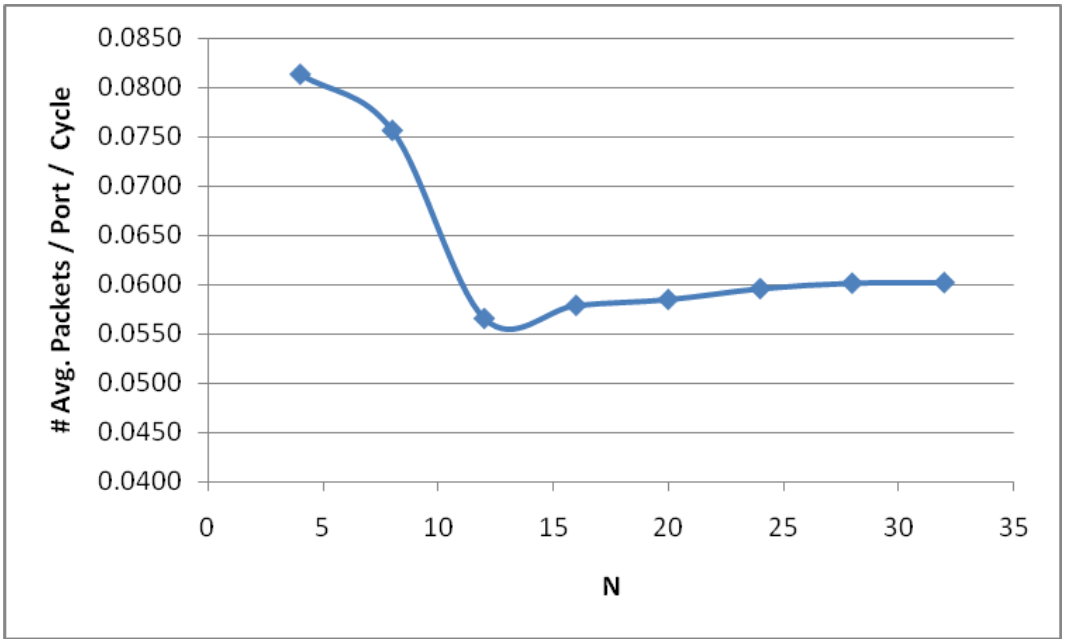


Figure 6.16: Average Number of Packets/Output Port/Cycle vs. N, after saturation

however, as the number of I/O ports increase, the traffic gets congested, decreasing the throughput. The increase after N = 16, can be explained with the fact that the number of routers becomes greater than the number of inputs, supporting higher throughput.

In order to compute the "Number of Packets / Cycle", the overall throughput of the switch, we multiply the values in Table 6.7 by N, and present the results in Table 6.8 and Figure 6.17. The graph increases linearly, as N is increased, which means that it scales well under the Bernoulli Uniform Traffic.

Table 6.8: Number of Packets/Cycle, after saturation

N	# Packets / Cycle
4	0.3253
8	0.6051
12	0.6790
16	0.9265
20	1.1700
24	1.4304
28	1.6846
32	1.9270

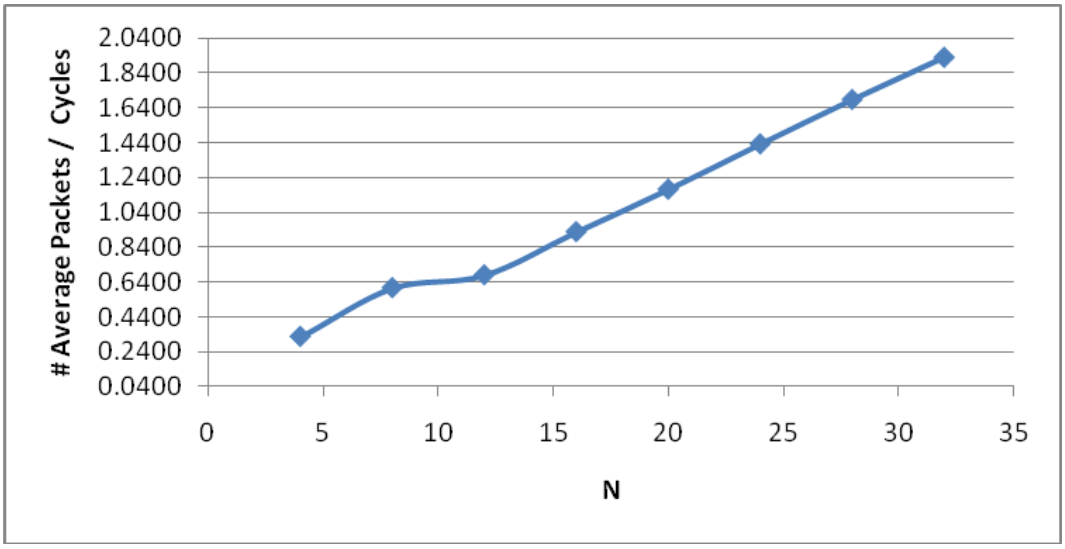


Figure 6.17: Number of Packets/Cycle vs. N, after saturation

6.3.2 Simulations under Unbalanced (Weighted) Traffic

The simulation results of MDN Switches with possible (N) values, under Unbalanced (Weighted) Traffic, are presented in Table 6.9 and its respective graph in Figure 6.18.

Table 6.9: Average Number of Packets/Output Port/Cycle, after saturation

N	# Avg. Packets / Port / Cycle
4	0.0634
8	0.0546
12	0.0511
16	0.0483
20	0.0452
24	0.0403
28	0.0344
32	0.0304

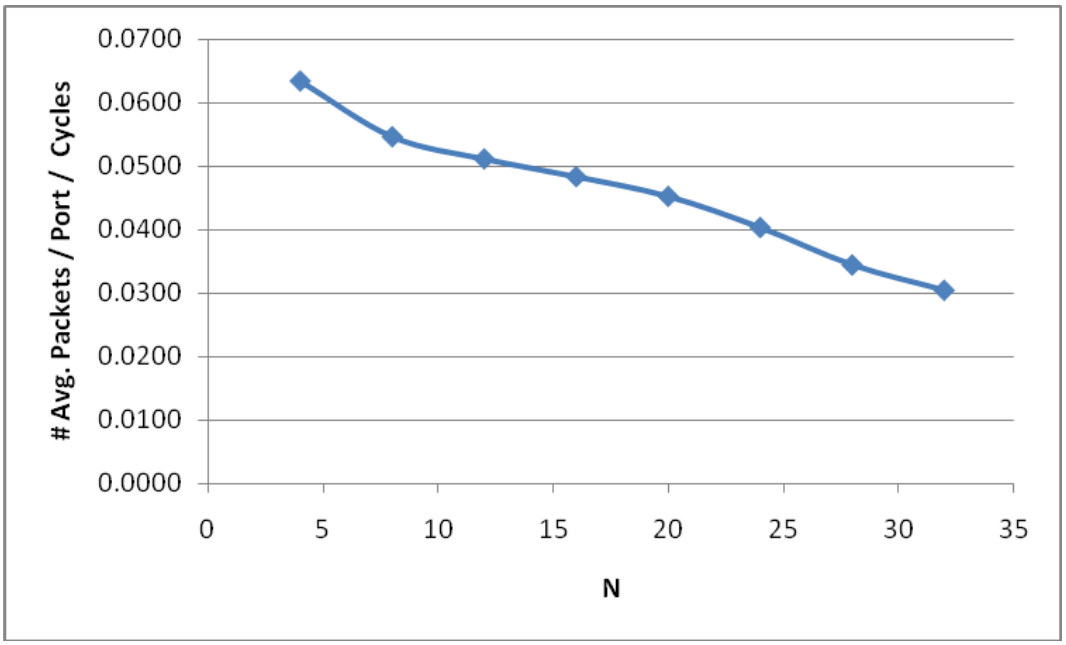


Figure 6.18: Average Number of Packets/Output Port/Cycle vs. N, after saturation

It is noteworthy to observe that MDN Switch performance per port decreases linearly under the Unbalanced Traffic, as N is increased. Comparing its performance to the MDN simulations under Bernoulli Uniform Traffic, it can be concluded that the switch cannot be scaled under Unbalanced Traffic.

In order to compute the "Number of Packets / Cycle", the overall throughput of the switch, we multiply the values in Table 6.9 by N, and present the results in Table 6.10

and Figure Figure 6.19. The graph increases linearly, until it saturates at a certain value. This also shows that the MDN Switch is not scalable under Unbalanced Traffic, since there is no more increase in throughput, as the number of I/O ports are increased.

Table 6.10: Average Number of Packets/Output Port/Cycle, after saturation

N	# Average Packets / Cycle
4	0.2535
8	0.4368
12	0.6132
16	0.7728
20	0.9040
24	0.9672
28	0.9632
32	0.9728

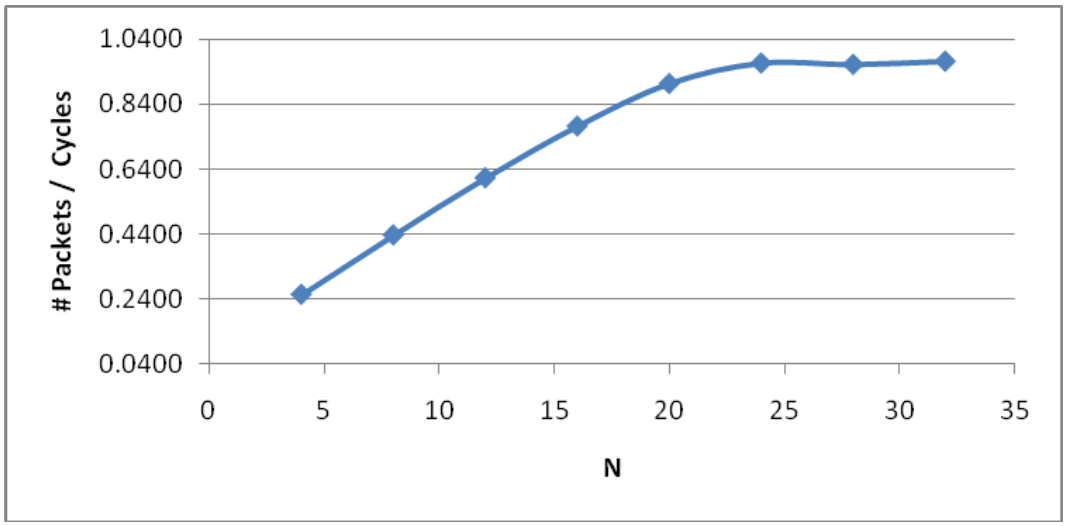


Figure 6.19: Average Number of Packets/Output Port/Cycle vs. N, after saturation

6.4 Comparison of Arbiter Schemes

In this section we present the simulations of the UDN arbiter schemes, as explained in Section 4.2.1. First we compare static and dynamic arbitration schemes, on UDN Switches (with reduced router types) of size N = 8, and M = (1, 2, 4, 7, 8), over uniform traffic. The performance comparison is given in Table 6.11/Figure 6.20, where the values are the average of 2500 cycles, after the values are saturated.

Table 6.11: Performance Comparison of Static and Dynamic Arbitration

	Static		Dynamic	
UDN Switch	Average Packets pp / Cycle	% Sent Packets	Average Packets pp / Cycle	% Sent Packets
(8,1)	0.0461	99.2607	0.0447	99.2239
(8,2)	0.0651	99.3627	0.0671	99.3731
(8,4)	0.0810	99.3406	0.0814	99.3681
(8,7)	0.0873	99.1847	0.0868	99.2069
(8,8)	0.0873	99.1344	0.0867	99.1561

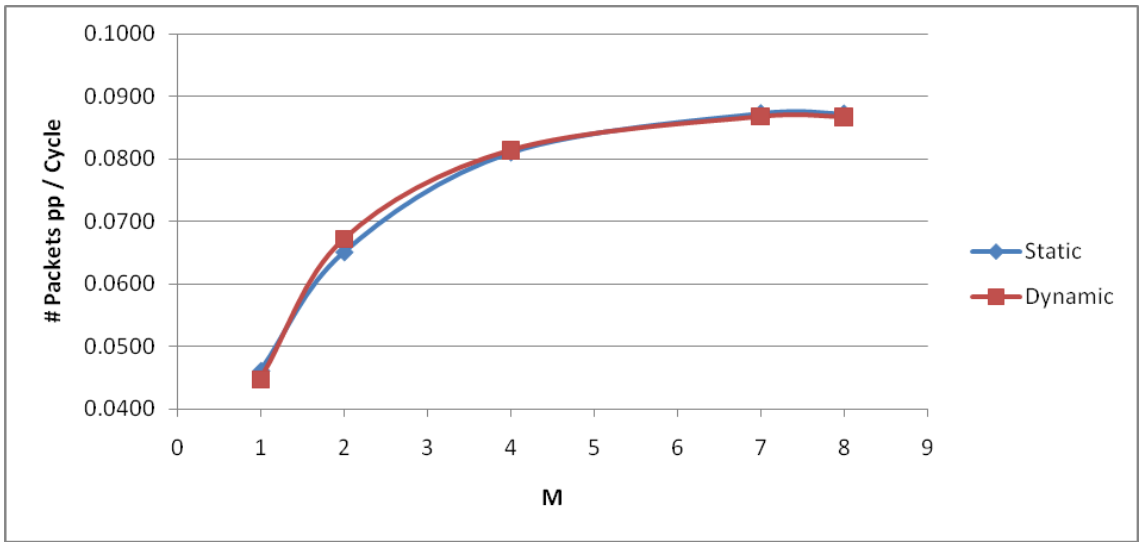


Figure 6.20: Performance Comparison of Static and Dynamic Arbitration

The results show that the Static and Dynamic arbitration schemes result in very similar results in UDN Switch. For some M values, the throughput is slightly higher with static arbiters; and for some other M values, vice versa. This means that for UDN Switch architecture, both of the arbitration schemes are applicable.

In order to compare Static arbitration with weighted arbitration, we return back to

our example in Section 3.5.1, the UDN Switch of size (3, 2) and designate three cases: 1) (3, 2) UDN Switch with reduced routers and static arbiter, 2) (3, 2) UDN Switch with unreduced routers and static arbitration, and 3) (3, 2) UDN Switch with unreduced routers and weighted arbitration. RRR values in the routers for the respective test cases are given below, where OtherPort constant is either South or North, according to the position of 2 I/O Port Router.

1)

2 I/O Port Router

East RRR	(West, OtherPort)
OtherPort RRR	(West)

3 I/O Port Router

East RRR	(West, South, North)
South RRR	(West, North))
North RRR	(West, South)

2)

2 I/O Port Router

East RRR	(West, OtherPort)
OtherPort RRR	(West)

3 I/O Port Router of Type 3

East RRR	(West, South)
South RRR	(North))
North RRR	(West)

3 I/O Port Router of Type 1

East RRR	(West, North)
South RRR	(West))
North RRR	(South)

3)

2 I/O Port Router

East RRR	(West, West, OtherPort)
OtherPort RRR	(West)

3 I/O Port Router of Type 3

East RRR	(West, West, South)
South RRR	(North))
North RRR	(West)

3 I/O Port Router of Type 1

East RRR	(West, West, North)
South RRR	(West))
North RRR	(South)

In Table 6.12, "Average Number of Packets / port / cycle" and "% Sent Packets" results are presented for all of the test cases. The second test case performs better than the first, because the unreduced routers are more efficient for the specific traffic flows on their respective positions in the UDN Switch. The third case performs much better than the others, since its RRRs have the weighted values, and therefore the arbiter performs better in granting permission to the input ports over the output port, according to the weights of the traffic flows.

In Section 3.5.1, we have explained that we have reduced the router types into two types, in order to increase the reusability and scalability of the RTL description of the implementation. While this holds to be a necessary decision implementation-wise for greater degree of reconfigurability, here we also show that it decreases the performance.

Table 6.12: Performance Comparison of Static and Weighted Arbitration

	Test Cases					
	1		2		3	
Switch Size	# Packets pp / Cycle	% Sent Packets	# Packets pp / Cycle	% Sent Packets	# Packets pp / Cycle	% Sent Packets
(3, 2)	0.092569	99.6532	0.093360	99.6568	0.096115	99.6518

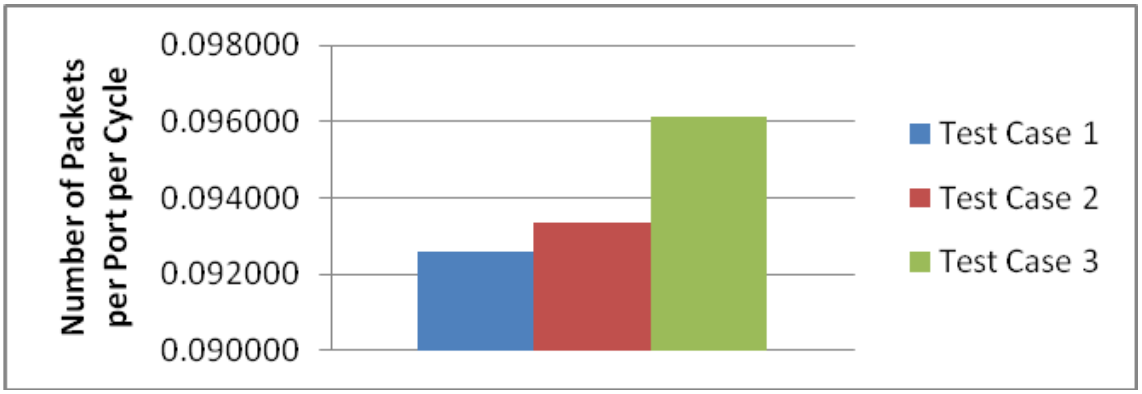


Figure 6.21: Performance Comparison of Static and Weighted Arbitration

6.5 Increasing Throughput with T-Value

We have explained that T-Value parameter can be used to balance the load on the switch, by changing the column the packets are routed through vertically for a certain Switch I/O Port pair. In this thesis, we did not implement a load observation unit, which would calculate T-Values according to the network state; however we investigate the effect of T-Values by assigning them random values and measuring the throughput of the switches, as a comparison to the case where all T-Values are set to 0. The simulations are carried on UDN Switches with size of (8,1), (8,2), (8,4), (8,7), (8,8). T-Values are randomly generated in the range of 0 and M-1. The simulation results are presented in Table 6.13 and Figure 6.22.

In all cases, the random T-Values result in equal or better performance than setting them to 0. This can be explained by the fact that the load balancing on the switch decreases the congestion over certain links, by providing better packet distribution on the columns, which improves the throughput performance.

In the following figures, we present the ratio of packet loads over the links of UDN Switch of size (8, 7), with T-Values set to 0. The Figure 6.23 shows the ratio of number

Table 6.13: Evaluation and comparison of two cases: T-Values=0 and T-Values=Random(0..M-1)

	T-Value = 0		T-Value = Random (0..M-1)	
UDN Switch Size	# Packets pp / Cycle	% Sent Packets	# Packets pp / Cycle	% Sent Packets
(8,1)	0.0461	99.2607	0.0460	99.2555
(8,2)	0.0651	99.3627	0.0730	99.4770
(8,4)	0.0813	99.3406	0.0874	99.4263
(8,7)	0.0875	99.1847	0.0889	99.2296
(8,8)	0.0875	99.1344	0.0916	99.2130

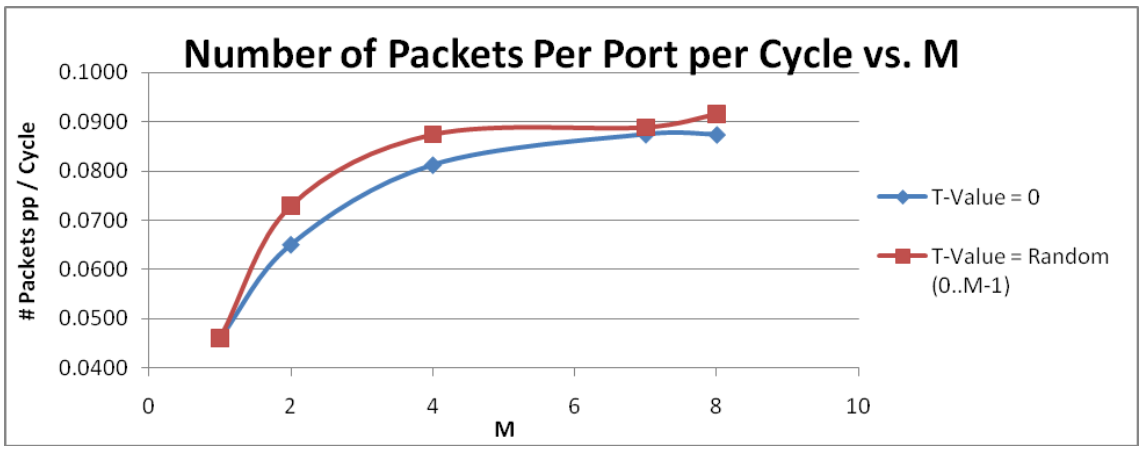


Figure 6.22: Evaluation and comparison of two cases: T-Values=0 and T-Values=Random(0..M-1)

of packets over the West ⇒ East links; the Figure 6.24, for South ⇒ North links, and Figure 6.25, for North ⇒ South links. In this way we can show the accumulative distribution of packets over the switch. The West ⇒ East links are loaded more around the eastmost and westmost columns, and less in the center; the South ⇒ North and North ⇒ South link, on the other hand, are loaded more in the center.

On the other hand, carrying out the same simulations with random T-Values, we can observe that the packet loads over the links are different from the case where T-Values are set to 0, in Figure 6.26 - Figure 6.28. The higher ratios exist only on a very small number of links, and the congestion in the central region of the switch is resolved for South ⇒ North and North ⇒ South links. Using the random T-Values, the vertical traffic is balanced and well distributed, resulting in greater throughput.

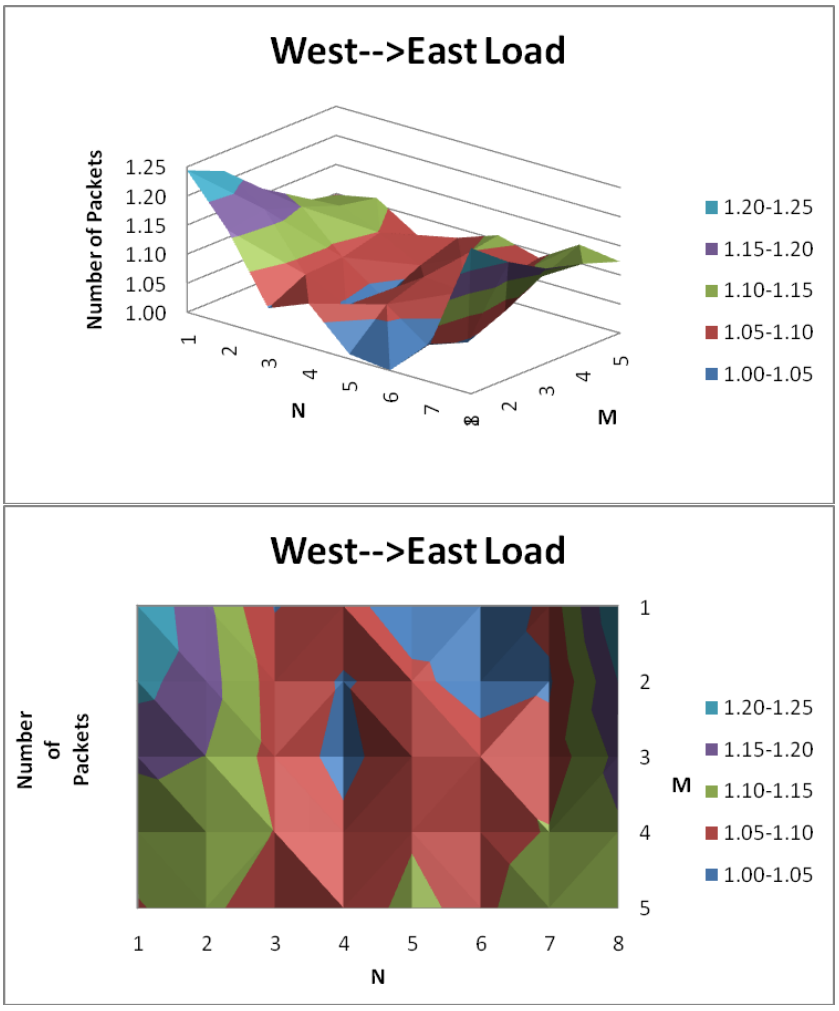


Figure 6.23: Packet Load over West to East Links

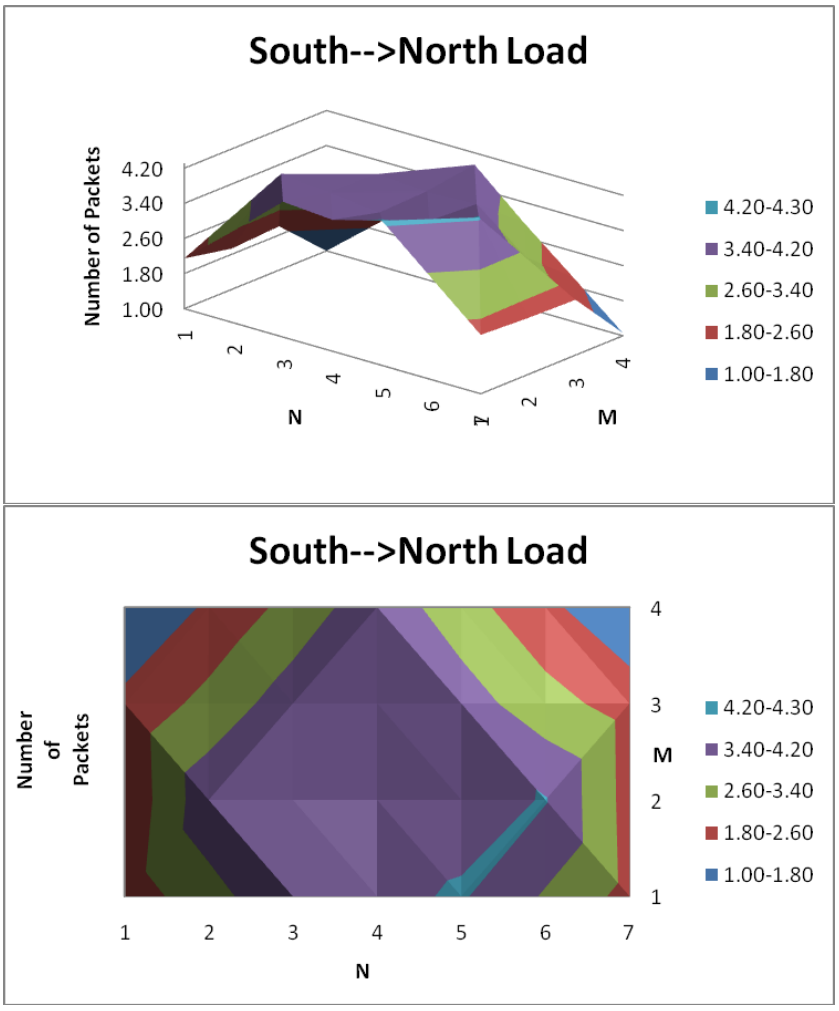


Figure 6.24: Packet Load over South to North Links

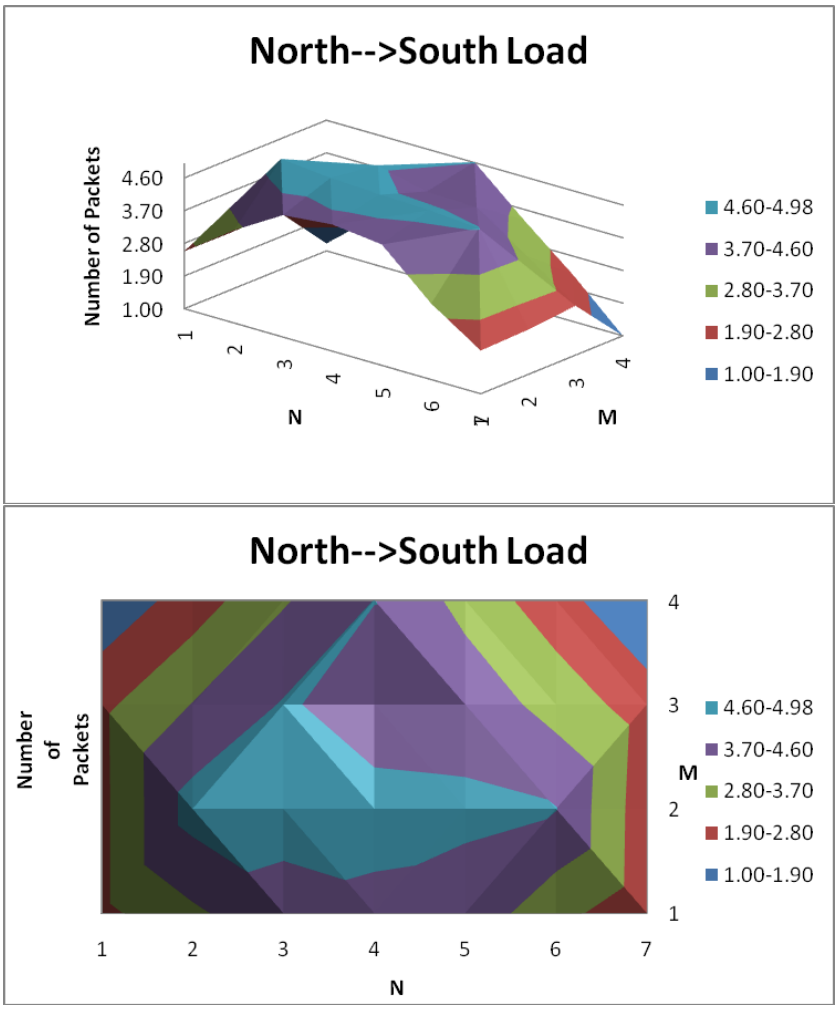


Figure 6.25: Packet Load over North to South Links

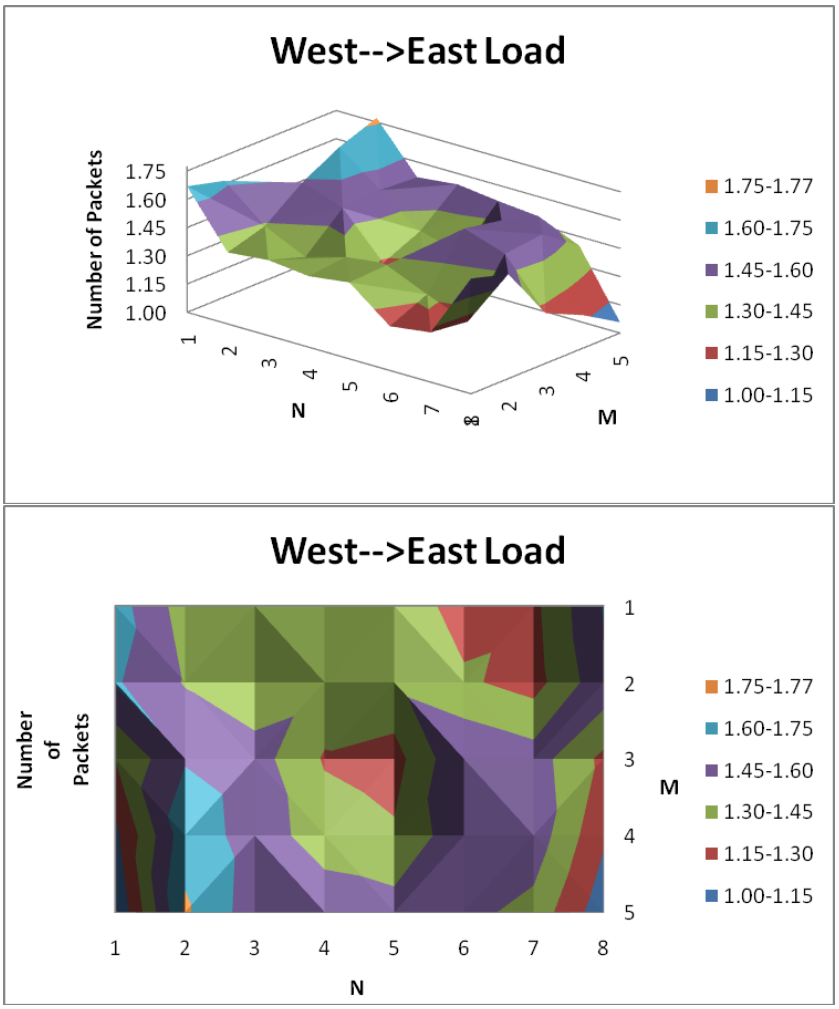


Figure 6.26: Packet Load over West to East Links

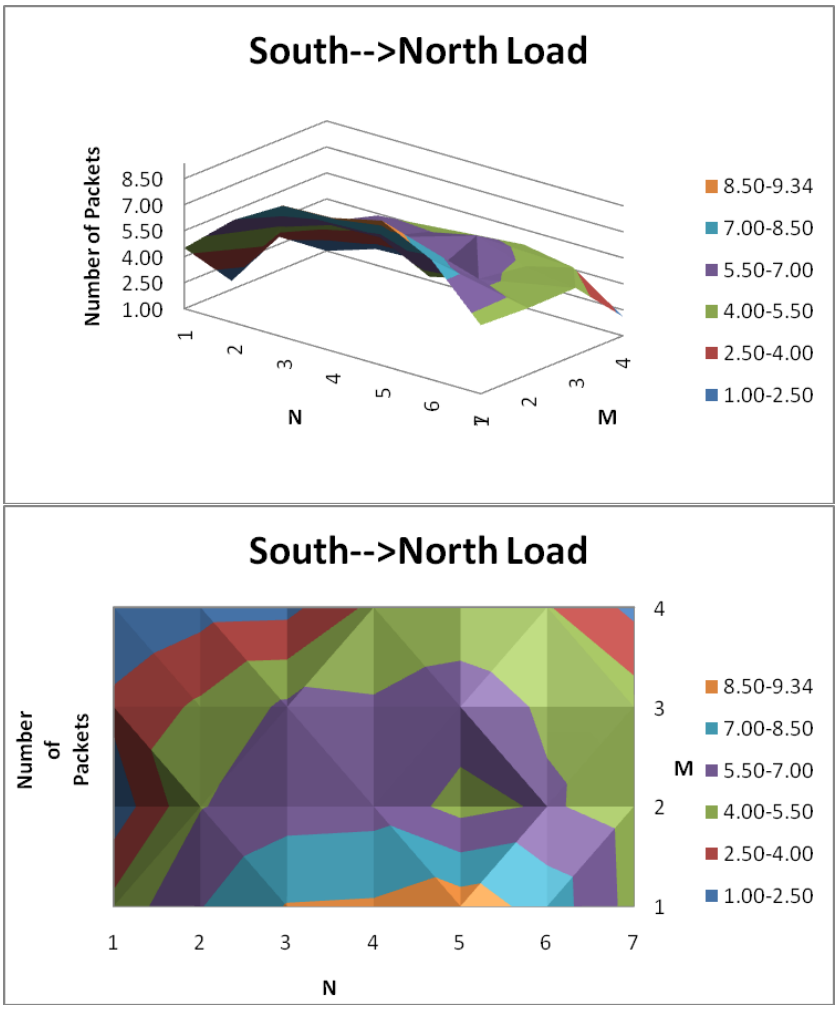


Figure 6.27: Packet Load over South to North Links

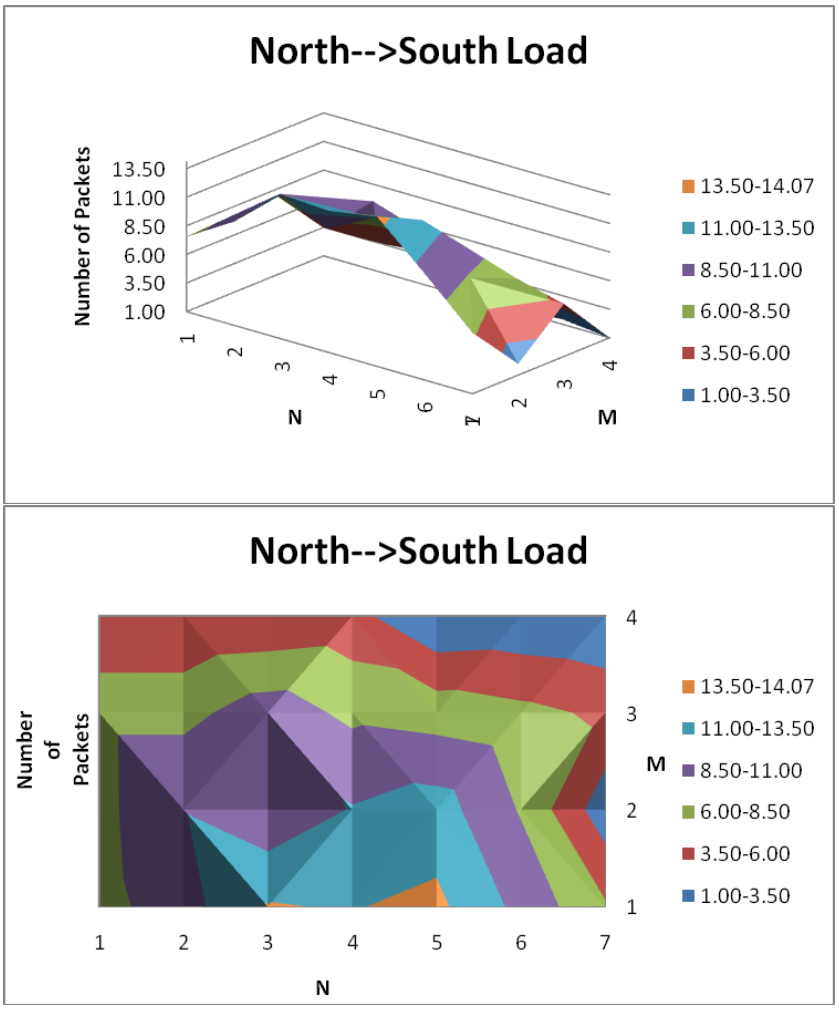


Figure 6.28: Packet Load over North to South Links

UDN/MDN Performance & Cost Analysis

7

In this chapter, we present the performance and cost analysis of the UDN and MDN switches. The performance and cost analysis are done with regard to the findings of the synthesis (Chapter 5) and simulations (Chapter 6) that were carried out with the static arbitration scheme, under Bernoulli Uniform Traffic and with T-Values set to 0.

In the first section, we define the performance and cost functions, and then present the respective data and analysis.

7.1 Defining the Performance & Cost Functions

7.1.1 Performance

We define the **UDN/MDN Performance** metric to be the data throughput, which is expressed as "Number of packets / time". "Number of packets / cycle" is measured in the simulations. The cycles can be converted in seconds by multiplying them with the operational frequency of the respective switch size, computed during synthesis.

UDN/MDN Performance = Throughput Performance
Throughput Performance = Number of Packets / Time
Number of Packets / Time = Number of Packets / Cycle $\times$ Operational Frequency

7.1.2 Cost

Since the implementation of the switch is made for FPGA platforms, we define the 2-tuple UDN/MDN Cost, in the same way XST reports it, as ("Number of Slice LUTs", "Number of Slice Registers"). These values are computed during the FPGA synthesis.

UDN/MDN Cost = ("Number of Slice LUTs", "Number of Slice Registers")

7.1.3 Performance / Cost

The 2-tuple Performance per Cost compound metric makes it possible to compare UDN and MDN switches, which offer different ratios of performance and cost.

UDN/MDN Performance per Cost = ("UDN/MDN Performance" / "Number of Slice LUTs", "UDN/MDN Performance" / "Number of Slice Registers")

7.2 UDN Performance & Cost Analysis

In Table 7.1 and Figure 7.1, we present the UDN Performance, for (N, M) sized switches. The values in this table are computed by multiplying the "Number of Packets/Cycle" values in Table 6.3 with the corresponding operational frequencies in Table 5.2. UDN Performance increases logarithmically with respect to M, which means that the highest performance is achieved when N=M.

Table 7.1: UDN Performance

N\M	1	2	3	4	7	8	15	16	31	32
2	62357342	62326418								
3	80037254	81570607								
4	96036729	97273391	102501291	102448898						
5	102537658	118753481		124496228						
8	104417272	147404068		183570920	197781641	197681380				
12	90320870	146244967		223317442		260299082				
16	83171516	143560788		244644824		335438895	362165304	361998032		
32	64845190	109970891		215152640		458070787		614341317	658173096	657841919

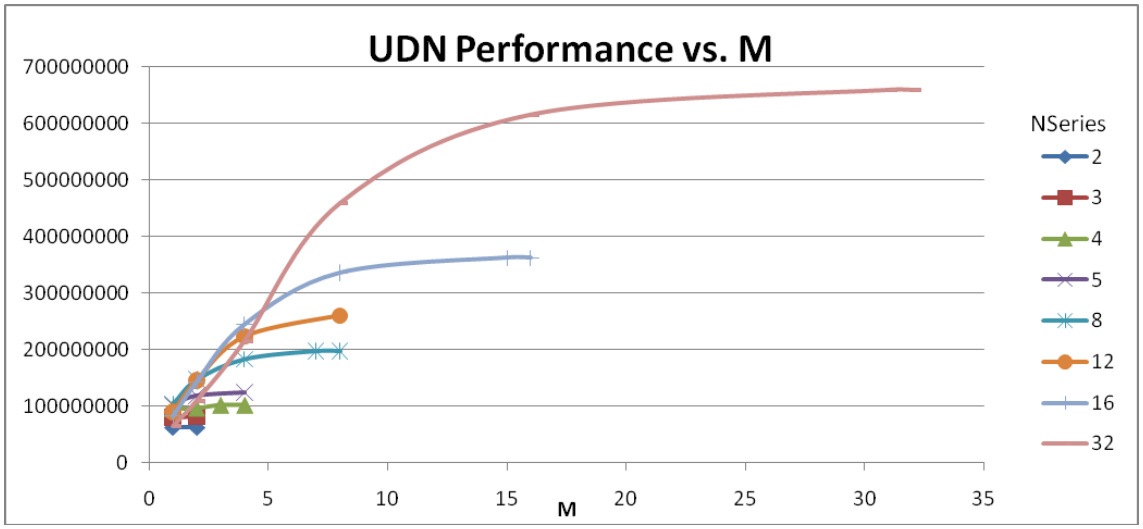


Figure 7.1: UDN Performance

UDN Cost results were presented in Chapter 5, as "Number of Slice LUTs" and "Number of Slice Register" on Table 5.3 and Table 5.4, respectively. Here, we present the same tables, for the reader's convenience, in Table 7.2 and Table 7.3

Table 7.2: Number of Slice LUTs for various (N, M) Switches

N\M	1	2	3	4	7	8	15	16	31	32
2	5148	6376								
3	9802	16796								
4	14415	24987	31759	37106						
5	18998	33783		62375						
8	32757	59140		109007	179978	184203				
12	51404	83915		157282		330422				
16	69883	116600		212202		412382	762421	815425		
32	127953	227833		428484		829785		1632388	3137269	3237595

Table 7.3: Number of Slice REGs for various (N, M) Switches

N/M	1	2	3	4	7	8	15	16	31	32
2	7516	9562								
3	12301	19467								
4	17089	27324	37590	41655						
5	21875	35184		61797						
8	36236	58771		103838	171508	179707				
12	55396	90251		159977		299236				
16	74545	121683		215954		404332	734114	781216		
32	151129	247409		439970		825090		1595332	3039532	3135810

The UDN Performance per Cost compound metric for (N, M) sized switches is pre-

sented in Table 7.4 (Figure 7.2) and Table 7.5 (Figure 7.3) respectively. Even though the overall performance of (32, 32) UDN switch is ≈ 10 times greater than (2, 1) UDN switch, the Performance per LUT Cost compound metric is ≈ 60 times smaller. This shows that UDN switch is cost inefficient, even though it provides high performance. The compound metric will be further used to compare the UDN switch to MDN, in the following sections.

Table 7.4: UDN Performance / Number of Slice LUTs Cost

N/M	1	2	3	4	7	8	15	16	31	32
2	12113	9775								
3	8165	4857								
4	6662	3893	3227	2761						
5	5397	3515		1996						
8	3188	2492		1684	1099	1073				
12	1757	1743		1420		788				
16	1190	1231		1153		813	475	444		
32	507	483		502		552		376	210	203

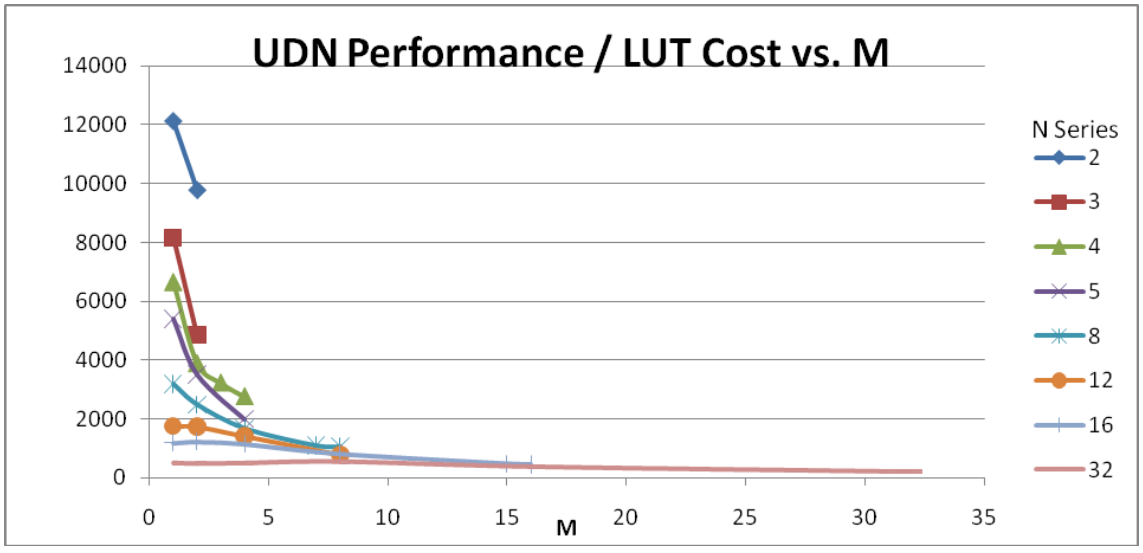


Figure 7.2: UDN Performance / Number of Slice LUTs vs. M

Table 7.5: UDN Performance / Number of Slice REGs Cost

N/M	1	2	3	4	7	8	15	16	31	32
2	8297	6518								
3	6507	4190								
4	5620	3560	2727	2459						
5	4687	3375		2015						
8	2882	2508		1768	1153	1100				
12	1630	1620		1396		870				
16	1116	1180		1133		830	493	463		
32	429	444		489		555		385	217	210

Figure 7.3: UDN Performance / Number of Slice Registers vs. M

7.3 MDN Performance & Cost Analysis

In Table 7.6 and Figure 7.4, we present the MDN Performance, for (N) sized switches, under uniform traffic. The values in this table are computed by multiplying the "Number of Packets/Cycle" values in Table 6.8, with the corresponding operational frequencies in Table 5.6. MDN Performance increases linearly with respect to the number of I/O ports, proving its scalability.

MDN Cost results were presented in Chapter 5, as "Number of Slice LUTs" and "Number of Slice Register" on Table 5.7 and Table 5.7, respectively. Here, we present

Table 7.6: MDN Performance

N	Throughput Performance
4	81512087
8	145617292
12	158609820
16	196625273
20	235174020
24	273427016
28	312607195
32	333775976

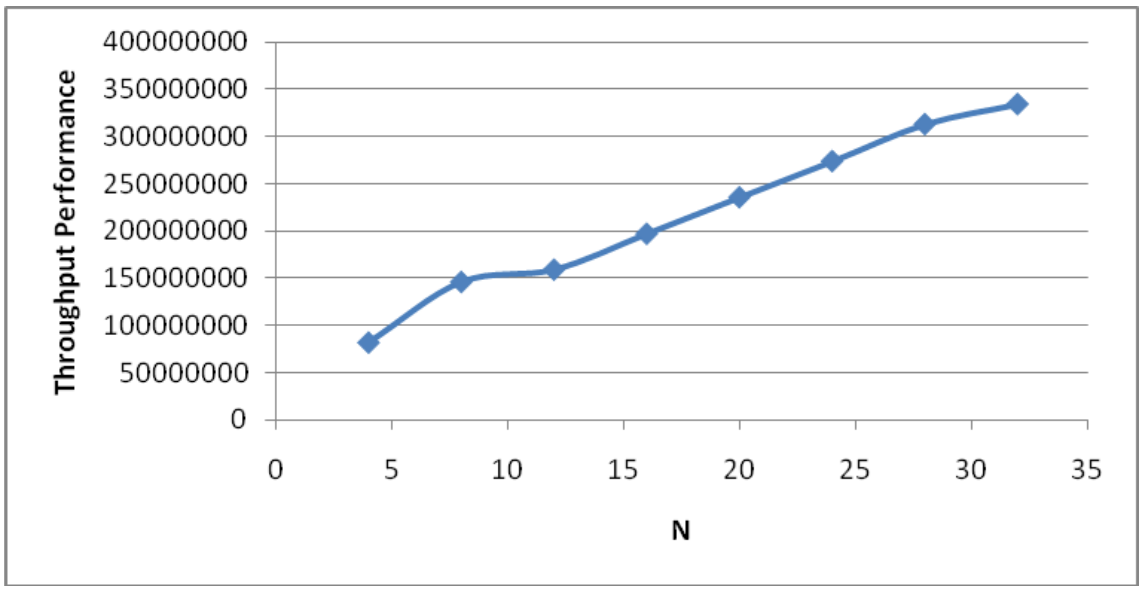


Figure 7.4: MDN Performance

the same tables, for the reader's convenience, in Table 7.7 and Table 7.8.

The MDN Performance per Cost compound metric for (N) sized switches is presented in Table 7.9 and Table 7.10, and their respective figures Figure 7.5 and Figure 7.6. Even though the overall performance of (32) MDN switch is ≈ 4 times greater than (2) MDN switch, the Performance per LUT Cost compound metric is ≈ 11.2 times smaller. This shows that higher performance through greater switch sizes is expensive for MDN switch.

Table 7.7: Number of Slice LUTs for various MDN Switches

N	Number of Slice LUTs
4	8933
8	30387
12	62678
16	108996
20	166373
24	229778
28	312343
32	410697

Table 7.8: Number of Slice Registers for various MDN Switches

N	Number of Slice Regs
4	10969
8	30153
12	57630
16	93203
20	137080
24	189168
28	249480
32	318016

Table 7.9: MDN Performance / Number of Slice LUTs Cost

N	Performance / # Slice LUTs
4	9125
8	4792
12	2531
16	1804
20	1414
24	1190
28	1001
32	813

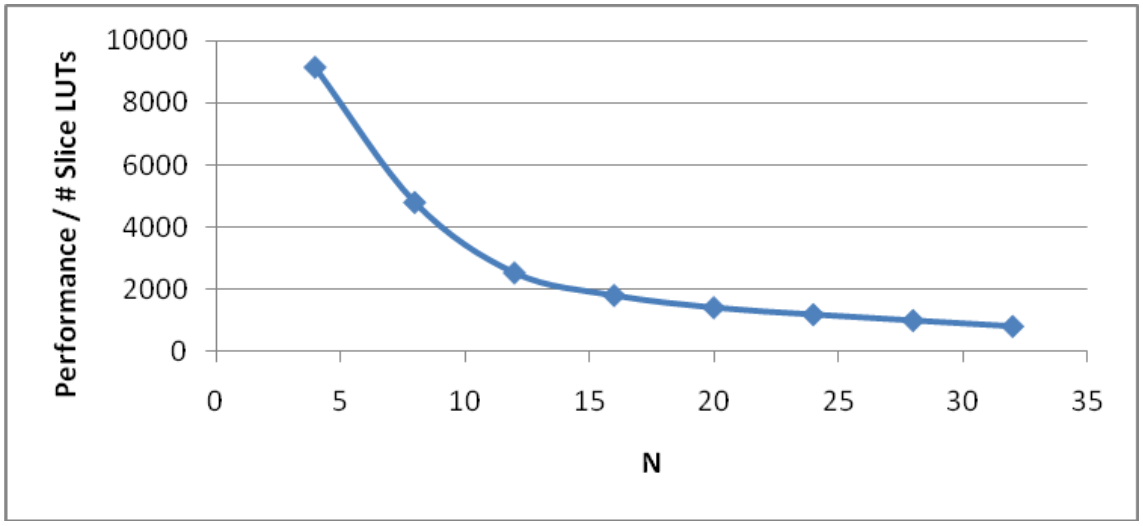


Figure 7.5: MDN Performance / Number of Slice LUTs vs. N

Table 7.10: MDN Performance / Number of Slice REGs Cost

N	Performance / # Slice REGs
4	7431
8	4829
12	2752
16	2110
20	1716
24	1445
28	1253
32	1050

7.4 UDN/MDN Comparison

In this section, we compare the UDN and MDN switches. We have chosen to compare four UDN and MDN switches with the same number of input/output ports. For UDN, we have chosem M = N - 1, since it yields the greatest performance compared to other M values. The comparisons are presented in Table 7.11 - Table 7.13.

UDN has higher operational frequency than the corresponding MDN switch. UDN has its critical path on the arbitration and flow control combinational cloud of its 3 I/O Port Router, whereas MDN has it on its 4 I/O Port Router, with more complex arbiter, resulting in longer critical path.

The UDN LUT and Register cost is much greater than MDN. The (8, 7), (16, 15), (32, 31) UDN switches require more than 100% of the resources on a Virtex-5 (XC5VTX240T,

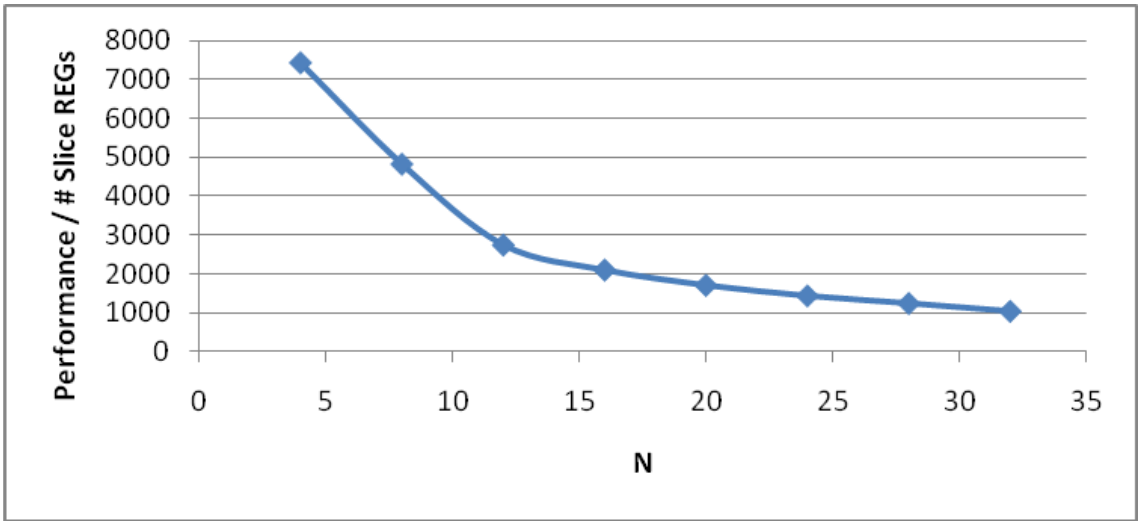


Figure 7.6: MDN Performance / Number of Slice Registers vs. N

FF1759, -2), and therefore cannot be placed on the FPGA. On the other hand, only (32) MDN switch cannot be placed on the FPGA.

Table 7.11: Comparison of Synthesis Results

Compared Switch Sizes		Frequencies		# of Slice LUTs		# of Slice REGs	
UDN	MDN	UDN	MDN	UDN	MDN	UDN	MDN
(4, 3)	4	290.252	250.591	31759	8933	37590	10969
(8,7)	8	283.168	240.667	179978	30387	171508	30153
(16, 15)	16	252.139	212.227	762421	108996	734114	93203
(32, 31)	32	220.146	173.214	3137269	410697	3039532	318016

In the simulation phase, it was observed that UDN performance is greater than MDN, since less number of ports competes for an output port in UDN Router (3, as opposed to 4), resulting in less contention and congestion, and therefore greater throughput.

Even though the throughput performance is greater for UDN, the performance/cost compound metrics show that MDN's performance per resource cost is much greater than UDN. This is because UDN switch resource costs increase quadratically with respect to $N \times M$ product, whereas it increases subquadratically ($(N/4)^2$) in MDN.

In order to make it easier to compare the UDN and MDN switches' performance to other switches, we also present the throughput performance in terms of Gigabytes/sec, in Table 7.14.

Table 7.12: Comparison of Simulation Results

Compared Switch Sizes		# Avg. Packets / Port / Cycle		# Avg. Packets / Cycle	
UDN	MDN	UDN	MDN	UDN	MDN
(4, 3)	4	0.08829	0.0813	0.35315	0.32528
(8,7)	8	0.08731	0.0756	0.69846	0.60506
(16, 15)	16	0.08977	0.0579	1.43637	0.92649
(32, 31)	32	0.09343	0.0602	2.98971	1.92696

Table 7.13: Comparison of Performance and Performance/Cost

Compared Switch Sizes		Throughput Performance (# packets/ sec)		Performance / # Slice LUTs		Performance / # Slice LUTs	
UDN	MDN	UDN	MDN	UDN	MDN	UDN	MDN
(4, 3)	4	102501291	81512087	3227	9125	2727	7431
(8,7)	8	197781641	145617292	1099	4792	1153	4829
(16, 15)	16	362165304	196625273	475	1804	493	2110
(32, 31)	32	658173096	333775976	210	813	217	1050

Table 7.14: Throughput Performance in Gbytes / sec

Compared Switch Sizes		Throughput Performance (# Packets / sec)		Throughput Performance (Gbytes / sec)	
UDN	MDN	UDN	MDN	UDN	MDN
(4, 3)	4	102501291	81512087	6.56	5.22
(8,7)	8	197781641	145617292	12.66	9.32
(16, 15)	16	362165304	196625273	23.18	12.58
(32, 31)	32	658173096	333775976	42.12	21.36

8 In-Circuit Verification

In this section we present the mapping of the implementation on the reconfigurable platform. The implementation was mapped on a Virtex-5 FPGA board, specifically on xc5vtx240t-2ff1759. The testing system was implemented in Xilinx's Embedded Development Kit 11.1 (EDK). In this section we also present the architectures that we have used to generate random packets for in-circuit testing.

8.1 LFSR (Linear Feedback Shift Registers) for Testing

We have implemented a simple LFSR module as a building block of the synthesizable ATM generator, in order to validate the circuit on the FPGA board. This module yields 1-bit random output, and therefore it should be instantiated as many as required by a certain register width. In order to ensure the same bit values do not synchronize, the modules are instantiated with different initial seed values. The resulting pseudo-random value is the output address for a packet, and the width is LOG2(N). The block diagram is presented in Figure 8.1.

LFSR generates a series of pseudo-random bits, according to the initial seed value at reset. For example if the seed value is 1101, the order of outputs will be as in Table 8.1.

As one might notice, the number of generated 1s is more than number of 0s, by a difference of one. However, in [44] it is discussed that in a random stream, the difference between the two sums will tend to grow progressively smaller in proportion to the length of the stream as the stream gets longer. In order to have a better pseudo-randomness,

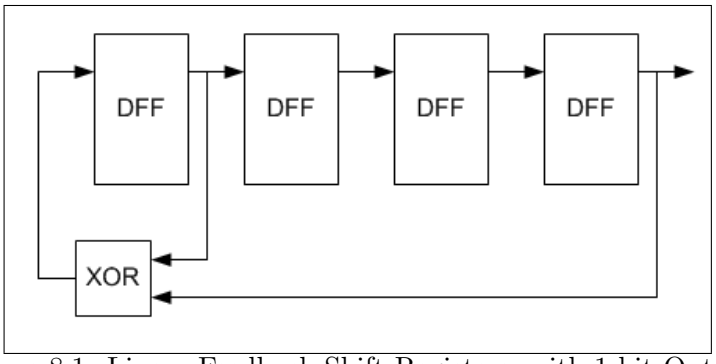


Figure 8.1: Linear Feedback Shift Registers, with 1-bit Output

Table 8.1: List of Generated Random Values on a 4-Bit LFSR, with Seed = 1101

cycles	FF	XOR	Output
0	1101	0	1
1	0110	0	0
2	0011	1	1
3	1001	0	1
4	0100	0	0
5	0010	0	0
6	0001	1	1
7	1000	1	0
8	1100	1	0
9	1110	1	0
10	1111	0	1
11	111	1	1
12	1011	0	1
13	0101	1	1
14	1010	1	0
15	1101	0	1

number of registers should be increased in LFSR. For the purpose of validating the design on FPGA, we leave the precision at 4 bits.

However, the solution above solves only the problem of random address generation for Bernoulli Uniform Traffic. For unbalanced traffic, we would need weighted random generation. Weighted random values can be achieved with weight rounding, in a simple way. This is implemented as an array of values, in which the values with higher weights are more frequent. This array is indexed with a variable, which is randomly generated by the LFSR. Since it is easier to generate index value in between 0 - 2^A, where A denotes number of randomly generated bits, the weight needs to be rounded to an integer multiple of (1 / 2^A). The greater A is, more accurate the rounding will be. We leave the weighted random generation as a future work.

8.2 FPGA Validation

The UDN switch (N, M) = (4, 3) has been validated on Virtex-4 FPGA Platform, specifically on ML41x(XC4VFX60-FF1152-11) board. Due to the fact the board offers only a 100 MHz clock maximally, the full frequency of the switch (290.252 MHz) cannot be exploited. The block diagram for the validation system is presented in Figure 8.2.

The top level module with ATM Generators and NoC switch has a state machine involving two states, IDLE and RUN. In the IDLE state, ATM generator modules are not enabled, therefore no packet is injected in the NoC switch. Once the PowerPC microprocessor is initiated and the testing software runs, the first address of BRAM

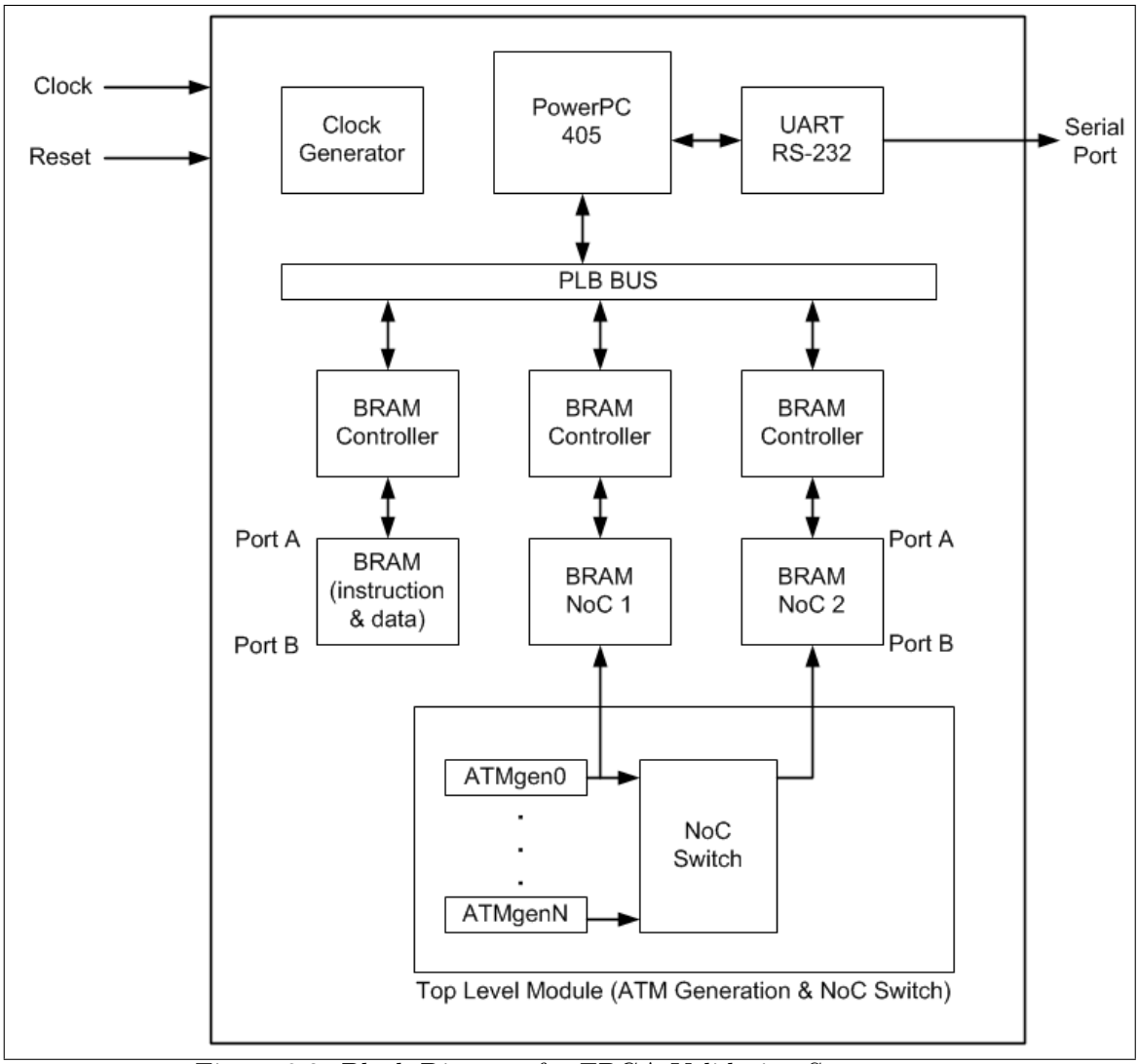


Figure 8.2: Block Diagram for FPGA Validation System

NoC 1 is set to 1. These trigger the ATM Generators and NoC switch, which changes the state machine from state IDLE to state RUN. As the input packets are injected to the NoC switch, they are also written to BRAM NoC 1. When the packets are transmitted to the destination address, they are written to BRAM NoC 2. In this way we can compare the packets, and validate if the system works correctly. With this approach, we were able to validate that all the packets have been transmitted correctly to the output ports and that the simulation results hold correct.

Using this system, we can also do some performance analysis. Once the system has been run a certain amount of time, the packet generation can be stopped and the NoC switch can be stalled by resetting the first address of BRAM NoC 1; in this way we ensure no new packets are transferred afterwards and therefore written to the memories.

Measuring the time period (in cycles) in between the points where the ATM generators were started and stopped, we can observe if the performance of the system on the FPGA platform matches the simulations. The pseudo-code example in Listing 8.1 describes this procedure.

Listing 8.1: Pseudo-code for Performance Analysis

```
//Set Memory Base Addresses
    bram1   = XPAR_BRAM_CNTLR_0_BASEADDR;
    bram2   = XPAR_BRAM_CNTLR_1_BASEADDR;
//Enable ATM generators and NoC Switch
    *bram1 = 1;
//Start PowerPC Assembly Timer
//Do some dummy operations that will take some time
//Disable ATM generators and Stall NoC Switch
    *bram1 = 0;
//Stop PowerPC Assembly Timer
//Compute number of input/output packets
//Compute the number of cycles the NoC Switch was active

//Compute Number of Output Packets per cycle
//Compute % Sent Packets (Output Packets/Input Packets)
```

Please note that the simulations are more accurate and precise compared to the performance analysis carried out on the FPGA board, due to a number of factors:

- Timing measurement is not very accurate; the number of cycles measured for the same code varies from one run to another. As a result, we run the code multiple times and averaged the number of cycles, which offers a remedy to the problem up to a certain degree.
- The LFSR generates seven pseudo random 0s, as opposed to eight 1s. This harms the generation of a uniform traffic as accurate as in the simulations.

In 9'723'460 cycles, the NoC switch is injected 3'159'577 packets and has ejected 3'144'095; then "Number of Output Packets per cycle" is equal to 0.32335 and "% Sent Packets" is equal to 99.51%, as opposed to 0.35315 and 99.541% values we have obtained in the simulations, respectively. The error percentage for the FPGA performance analysis results are 9.2% and 0.03%.

9 Conclusion

Recently, the functional-level design of two novel Network-on-Chip based switch fabric architectures were proposed, Unidirectional NoC (UDN) and Multidirectional NoC (MDN), as a replacement of the buffered crossbar switch fabric architecture [1][10][11].

The buffered crossbar switch consists of buffered crosspoints that makes point-to-point connections in between all input-output pairs. On the other hand, the NoC based crossbar switch consists of NoC routers, which are used to switch the packets from any input of the switch to any output. NoC routers are shared resources, unlike the dedicated crosspoints and the internal buffers. NoC also provides path diversity, as a result of multiple routing paths in between an input/output pair; this feature can be exploited for better load balancing over the switch fabric. The novel architectures based on NoC have many advantages over the internally buffered crossbar switches, including scalability, flexibility and reconfigurability.

In this thesis, we advanced the proposal in [1], by carrying out the hardware design and implementation of the aforementioned NoC based crossbar switches and studying their respective performance and cost metrics, over a wide range of switch sizes and traffic scenarios, in order to prove the claims of the original proposal. Moreover, we have implemented a test system in order to validate the switches on the reconfigurable FPGA platform and carry out some basic performance measurements.

Because of the non-uniform nature of UDN switch routers and traffic flows, as opposed to MDN, we have written a software simulator in order to investigate different types of routers, traffic flows and their frequencies on the UDN switch. This has led to our proposal on the Weighted Arbitration scheme, which proved to improve the throughput performance by 5% on a (3, 2) UDN Switch. However, we have decided to exchange this small performance increase, with flexibility and easy reconfigurability, by reducing the eight types of UDN Router into two types, and therefore onitting the use of Weighted Arbitration.

We have investigated load balancing and its effects on throughput performance, by comparing the test cases when T-Values are set to 0 and set to random values, and showing that random values perform better, with up to 12% speedup for certain switch sizes.

Through the simulations on UDN and MDN switches, under Bernoulli Uniform Traffic and Unbalanced (Weighted) Traffic, we have shown that the switches are performance-wise scalable, as the number of input/output ports are increased. UDN switch proved to

be scalable under both traffic types, whereas MDN failed scalability under Unbalanced Traffic.

We have shown that the UDN switch performs much better than MDN, in exchange of its high costs. By carrying out performance and cost analysis, we have shown that MDN yields better performance per cost. UDN offers 6.56 and 42.12 Gbytes/sec aggregate bandwidth for (4, 3) and (32, 31) switch sizes; which implies 1.64 and 1.32 Gbytes/sec bandwidth per port, respectively. On the other hand, MDN offers 5.22 and 21.36 Gbytes/sec aggregate bandwidth for (4) and (32) switch sizes; which implies 1.31 and 0.67 Gbytes/sec bandwidth per port, respectively. High performance Ethernet cables offer 1.25 Gbytes/sec of bandwidth; therefore, UDN switch proves to be a competitive architecture to comply with the market products, whereas MDNs performance does not match this bandwidth, as number of input/output ports is increased. UDN's high performance makes it suitable for performance critical cases, whereas MDN is a better solution for cases that require cost efficiency.

All in all, the claims in the original proposal [1] are proved to be correct, in terms of the NoC based switches' high-performance, scalability and flexibility.

Implementing of a load observation unit, to set the T-Values according to the state of the NoC switch, would implement adaptive routing and could improve the throughput and load balancing further. Implementing multicast traffic would provide further capability to the NoC switches, to route ATM packets with multiple destination addresses. In addition, UDN and MDN NoC switches would benefit from fault tolerance property, which can be implemented by dynamically reconfigurable UDN and MDN routers, which would be reconfigured to route the packets to an alternative path, in the case of a malfunctioning router. These are all left as future work, to improve the systems' performance, reliability and service capability.

Bibliography

[1] I. Senin, "Design of a High-Performance buffered crossbar switch fabric using network on chip," Master's thesis, TU Delft, MSc Thesis, TU Delft, 2008.

[2] D. E. McDysan and D. L. Spohn, *ATM theory and application*. McGraw-Hill, 1999.

[3] P. P. Chu, "Register based synchronous FIFO buffer," in *RTL hardware design using VHDL*, pp. 279–282, John Wiley and Sons, May 2006.

[4] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, *The Tiny Tera: A Packet Switch Core*. 1996.

[5] D. Torres, J. Gonzalez, M. Guzman, and L. Nunez, "A new bus assignment in a designed shared bus switch fabric," in *Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on*, vol. 1, pp. 423–426 vol.1, 1999.

[6] H. Kuwahara, N. Endo, M. Ogino, T. Kozaki, Y. Sakurai, and S. Gohara, "A shared buffer memory switch for an ATM exchange," in *Communications, 1989. ICC '89, BOSTONICC/89. Conference record. 'World Prosperity Through Communications', IEEE International Conference on*, pp. 118–122 vol.1, 1989.

[7] K. Genda, N. Yamanaka, Y. Doi, and K. Endo, "A 160 GB/s ATM switch using internal speed-up crossbar switch architecture," *Electronics and Communications in Japan (Part I: Communications)*, vol. 80, no. 9, pp. 68–79, 1997.

[8] D. Simos, I. Papaefstathiou, and M. Katevenis, "Building an FoC using large, buffered crossbar cores," *Design & Test of Computers, IEEE*, vol. 25, no. 6, pp. 538–548, 2008.

[9] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos, "Variable packet size buffered crossbar (CICQ) switches," in *Communications, 2004 IEEE International Conference on*, vol. 2, pp. 1090–1096 Vol.2, 2004.

[10] K. Goossens, L. Mhamdi, and I. Senin, "Internet-Router buffered crossbars based on networks on chip," in *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*, pp. 365–374, 2009.

[11] I. Senin, L. Mhamdi, and K. Goossens, "Efficient multicast support in buffered crossbars using networks on chip," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pp. 1–7, 2009.

[12] G. B. Shelly and M. E. Vermaat, *Discovering Computers: Fundamentals*. Cengage Learning, Mar. 2009.

[13] J. Aweya, "On the design of IP routers part 1: Router architectures," *Journal of Systems Architecture*, vol. 46, pp. 483–511, Apr. 2000.

[14] H. J. Chao and B. Liu, *High performance switches and routers.* Wiley-Interscience, Mar. 2007.

[15] L. Mhamdi, "Pbc: A partially buffered crossbar packet switch," *Computers, IEEE Transactions on*, vol. 58, pp. 1568 –1581, nov. 2009.

[16] K. Pun and M. Hamdi, "Distro: a distributed static round-robin scheduling algorithm for bufferless Clos-Network switches," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 3, pp. 2298–2302 vol.3, 2002.

[17] K. Goossens, "Network-on-Chip ET4361, TU delft [Lecture notes]."

[18] D. Flynn, "AMBA: enabling reusable on-chip designs," *Micro, IEEE*, vol. 17, no. 4, pp. 20–27, 1997.

[19] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 12, no. 3, p. 23, 2007.

[20] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv.*, vol. 38, no. 1, p. 1, 2006.

[21] H. Tenhunen and A. Jantsch, *Networks on Chip.* Springer, 1 ed., Jan. 2003.

[22] B. Vermeulen, J. Dielissen, K. Goossens, and C. Ciordas, "Bringing communication networks on a chip: test and verification implications," *Communications Magazine, IEEE*, vol. 41, no. 9, pp. 74–81, 2003.

[23] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Design Automation Conference, 2001. Proceedings*, pp. 684–689, 2001.

[24] A. Gelman, H. Kobrinski, L. Smoot, S. Weinstein, M. Fortier, and D. Lemay, "A store-and-forward architecture for video-on-demand service," in *Communications, 1991. ICC '91, Conference Record. IEEE International Conference on*, pp. 842–846 vol.2, 1991.

[25] T. Bartic, J. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins, "Highly scalable network on chip for reconfigurable systems," in *System-on-Chip, 2003. Proceedings. International Symposium on*, pp. 79–82, 2003.

[26] Z. Lu and A. Jantsch, "Admitting and ejecting flits in wormhole-switched networks on chip," *Computers & Digital Techniques, IET*, vol. 1, no. 5, pp. 546–556, 2007.

[27] D. Pao and S. Leung, "Sharing buffer in an input-output buffered ATM switch without scaling up memory bandwidth requirement," in *Computer Communications and Networks, 1995. Proceedings., Fourth International Conference on*, pp. 339–343, 1995.

[28] A. Hung, G. Kesidis, and N. McKeown, "ATM input-buffered switches with the guaranteed-rate property," in *Computers and Communications, 1998. ISCC '98. Proceedings. Third IEEE Symposium on*, pp. 331–335, 1998.

[29] G. J. Jeong, J. Lee, and B. Lee, "Implementation of high performance buffer manager for an advanced input-queued switch fabric," in *ASIC, 2002. Proceedings. 2002 IEEE Asia-Pacific Conference on*, pp. 37–40, 2002.

[30] W. Dally and C. Seitz, "Deadlock-Free message routing in multiprocessor interconnection networks," *Computers, IEEE Transactions on*, vol. C-36, no. 5, pp. 547–553, 1987.

[31] G. Smit, P. J. M. Havinga, and W. H. Tibboel, *Virtual lines, a deadlock free and real-time routing mechanism for ATM networks.* 1993.

[32] K. Seman, M. Waqas, and E. Kai, "Delay analysis of an input buffered ATM switch under two different scheduling disciplines," in *TENCON 2000. Proceedings*, vol. 3, pp. 266–270 vol.3, 2000.

[33] I. Radusinovic, M. Pejanovic, and Z. Petrovic, "iLPFwTM cell scheduling algorithm for ATM input-queued switch with service class priority," in *Electrotechnical Conference, 2002. MELECON 2002. 11th Mediterranean*, pp. 269–273, 2002.

[34] N. Matsurfuru and R. Aibara, "Efficient fair queueing for ATM networks using uniform round robin," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, pp. 389–397 vol.1, 1999.

[35] W. J. Dally and B. Towles, *Principles and practices of interconnection networks.* Morgan Kaufmann, 2004.

[36] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: concepts, architectures, and implementations," *Design & Test of Computers, IEEE*, vol. 22, no. 5, pp. 414–421, 2005.

[37] 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 *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2, pp. 890–895 Vol.2, 2004.

[38] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "Xpipes: a latency insensitive parameterized network-on-chip architecture for multiprocessor SoCs," in *Computer Design, 2003. Proceedings. 21st International Conference on*, pp. 536–539, 2003.

[39] T. Mattson, R. V. der Wijngaart, and M. Frumkin, "Programming the intel 80-core network-on-a-chip terascale processor," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pp. 1–11, 2008.

[40] T. Bjerregaard and J. Sparso, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Design, Automation and Test in Europe, 2005. Proceedings*, pp. 1226–1231 Vol. 2, 2005.

[41] "The ISO OSI seven layer model for networking." http://standards.iso.org/ittf/licence.html.

[42] "RFC1577 - classical IP and ARP over ATM."

[43] "RFC 2684 - multiprotocol encapsulation over ATM adaptation layer 5."

[44] S. W. Golomb, *Shift Register Sequences*. Aegean Park Pr, revised ed., June 1981.