IMPLEMENTATION AND EVALUATION OF LARGE INTERCONNECTION ROUTERS FOR FUTURE MANY-CORE NETWORKS ON CHIP

By
Amir Hasan Mohamed Zaytoun

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS ENGINEERING

Under the Supervision of

Prof. Khaled Mohamed Fouad Elsayed
Electronics and Communication Engineering

Assoc. Prof. Hossam Aly Hassan Fahmy
Electronics and Communication Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA EGYPT
2012

IMPLEMENTATION AND EVALUATION OF LARGE INTERCONNECTION ROUTERS FOR FUTURE MANY-CORE NETWORKS ON CHIP

By
Amir Hasan Mohamed Zaytoun

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS ENGINEERING

Approved by the Engineering Committee

_____

Prof. Dr. Khaled Mohamed Fouad Elsayed Thesis Main Advisor

_____

Assoc. Prof. Dr. Hossam Aly Hassan Fahmy Thesis Main Advisor

_____

Prof. Dr. Serag Eldin Elsayed Habib

_____

Assoc. Prof. Dr. Mohamed Watheq El-kharashi Associate Professor, Computer and Systems Engineering Department, Faculty of Engineering, Ain Shams University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA EGYPT
2012

Abstract

As the number of processing elements in the future Networks-on-Chip (NoC) increases from multi-cores to many-cores, the role of the interconnection communications becomes more critical. The number of cores on a System-on-Chip (SoC) will reach thousands in the near future as predicted by the International Technology Roadmap for Semiconductors (ITRS). Currently, NoC interconnections are mostly implemented with $m \times n$ 2-D mesh topology connecting small size routers. This will represent a bottleneck to the communication latency due the increasing number of cores, where the average number of hops the data have to pass will increase. In this thesis, we propose replacing the traditional mesh interconnecting scheme by using large routers interconnecting large number of cores in star topology. This interconnection scheme can be scaled up by using hierarchical-star or fat-tree topologies. We present the implementation and performance evaluation of three different architectures of 128×128 routers, the first is first-in-first-out input with Batcher-Banyan switching core, the second is virtual output queuing input with Batcher-Banyan switching core, the third is virtual output queuing input with crossbar switching core. Then, we compare their efficiency to the small 5×5 router used in the mesh topology. We develop a simulating environment using VHDL that resembles the real NoC conditions to test the routers throughput and average latency on different buffer sizes and under different traffic loads using ModelSim. We also synthesize them targeting to estimate the area and power consumption. Then, routers efficiencies are calculated with respect to the area and power consumption.

Table of Contents

This Page Intentionally Left Blank

Table of Figures

Abbreviations

ASIC	Application-Specific Integrated circuit
ASSP	Application-Specific Standard Products
CMOS	Complementary Metal Oxide Semiconductor
COTS	Commercial Off-The-Shelf
CSI	Cell Switch Interface
DSM	Deep Sub-Micron
DPM	Dynamic Power Management
DSP	Digital Signal Processor
Flit	Flow Control Digit
HOL	Head of Line
IC	Integrated Circuits
IP	Intellectual Property
NI	Network Interface
NoC	Networks-on-Chip
PCB	Printed Circuit Board
Phit	Physical Transfer Digit
RHE	Ring Head End
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SoC	System-on-Chip
TTM	Time To Market
UDSM	Ultra Deep Sub-Micron
VLIW	Very Long Instruction Word
VLSI	Very Large Scale Integration
VOQ	Virtual Output Queuing

This Page Intentionally Left Blank

Chapter 1

Introduction and Outline

1.1 Introduction

During the last decade, employing multi-core processors added a noticeable enhancement in performance and power consumption to electronic computing devices. The trend of increasing the number of processors gained more approbation than increasing the frequency and cache memory for single processor. The most famous example is Intel Core Duo, a dual-core processor on one chip released in January 2006 and Intel Core 2 Quad, a quad-core processor released in August 2008 [1].

A recent example of the trend of using distributed processing and decreasing operating frequency has been shown with the Polaris test chip in the Tera-Scale [2] research initiative proposed by Intel, where eighty simple cores are connected by using a bidirectional mesh. This chip, while still unable to execute complex control structures, is capable of delivering teraflop-level performance consuming less than 100 Watts [2]. IBM issued Cell Broadband Engine (CBE) in 2005. It consists of one 64-bit POWER™ Processing Element (PPE) and eight Synergistic Processing Elements (SPEs) [4].

The trend of increasing the number of processors is not of recent date, it first started by the end of the 20th century. Sun Microsystems designed since the mid-to-late 1990s the MAJC (Microprocessor Architecture for Java Computing), it was multi-core (two cores), multithreaded, very long instruction word (VLIW) microprocessor [5].

Commercial multiprocessor chips may contain few processors like Oracle SPARC T4 which is an example for the change in the design of general purpose

processors where it contains 8 cores and operate at 2.58 GHz and 3.0 GHz [6]. They may also contain a huge number of cores like Ambric Am2045 [7], which is 336 cores, with each core containing one (32-bit) RISC-DSP processor and one (2-KB) memory and running at up to 350 MHz.

In the field of graphic processing, Nvidia issued GeForce GTX 280 Graphics Cards on June 16, 2008 containing 240 processing cores. NVIDIA also issued Tesla C2075 which has 448 cores on one chip [8].

The technology of increasing the number of cores is the state of art now and is supposed to continue also in the next years.

1.2 Advances in CMOS Circuits

Gordon Moore, a co-founder of Intel Corporation, stated in 1965 that the number of components that could be incorporated per integrated circuit would increase exponentially over time. This statement is a famous formula called the Moore's law. Moore's law describes the number of transistors that can be placed on an IC. It doubles every 18 months. This would result in a reduction in the relative manufacturing cost per function, enabling the production of more complex circuits on a single semiconductor substrate. Along the past five decades, Moore's law was highly applicable in predicting chip capacity. So, the chip manufacturers became able to increase the number of transistors on one chip till it reached one billion [9][10][11][12]. According to the International Technology Roadmap of Semiconductors (ITRS), chips will grow to 4 billion transistors running around 10GHz by the end of the decade [13].

The increased demand on electronic devices led to the need for larger integrated circuits. Hence, the advancements in chip manufacturing processes forced fabrication engineers to shrink the size of transistors on silicon to make room for implementing hundreds of modules on a single chip. Hence,

emergence of Deep sub-micron (DSM) and Ultra DSM (UDSM) expressions, where semiconductor geometry shrinks below 350 nm and 100 nm, respectively.

Before DSM, the bottleneck of designing successful systems was the correct functionality of the system logic. There was no communication problem because gate delay on the chip dominated the wire delay. At the present transistor geometry (below DSM), the relation between gate delay and wire delay became different [14][50]. Gate delay decreased due to the decrease of parasitic capacitance of transistors because of the scaled down dimensions, where the gate length became smaller. On the other hand, metal wires also had to be decreased with the reduction in transistor geometry. The decrease of wire cross section area increased its resistance where $R = \frac{\rho L}{A}$, ρ is the metal resistivity, L is the wire length, and A is the wire cross section area. Increased wire resistance causes the data voltage swing to drop while passing on long buses, which affects the data integrity (correctness of data transmission). To solve the degradation in data voltage switching, we have to raise the operating voltage or insert buffers on the long buses. Increasing operating voltage definitely raises the power consumption ($P \alpha CV^2 f$ where P is the dynamic power, C is the capacitance of both wires and input load gates, V is the voltage swing, and f is the operating frequency). Inserting bus buffers raises the bus capacitance which also raises the power consumption.

On the other hand, the decreased transistor geometry in the present DSM and UDSM decreased its capacitance which hence allowed the ability to increase the operating frequency. So, interconnection delay now dominates over gate delay. At future designs of $2cm \times 2cm$ chips, a long diagonal global wire, which is the bus connecting system modules, causes a delay of 100ps, which is a full period of 10 GHz clock [15]. So, the interconnection between modules became a bottleneck in chip design and solutions have to be found.

Increasing the operating frequency to get more performance (frequency scaling) also reached the bottleneck point, where dynamic power dissipation in logic gates and wires raised to unacceptable levels. An example of this problem is Intel Pentium processors, where its operating frequency scaled till 3.6 GHz in Pentium 4 with more than 100W power consumption [1].

The solution of the aforementioned problems was in both lowering operating frequencies with distributing processing cores. Instead of one large processor with big cache memory and high frequency, we can distribute the work on small processors operating on lower frequency. The increasing demand for higher processing performance led to increasing number of processing cores on a single chip, the number of cores is expected to exceed one thousand cores in the future [2][16].

On the other hand, Application Specific Integrated Circuits (ASIC); which are a collection of function units designed for one product; and Application Specific Standard Products (ASSP); which are a collection of function units which can be used for a wide range of manufacturers and market products, have to contain many different processing units and peripherals to get more processing capability. Cell communication and automotive chips is a clear example of Chip Multi Processor (CMP), which is an example of ASSP.

The proposed solution of lowering the operating frequency with distributing the processing elements led to the possibility of integrating other various components on one chip. Mixing different modules on one chip reduced the dimensions of the Printed Circuit Boards (PCB). Instead of multiple chips, one chip can contain collection of general purpose processors, digital signal processors (DSP), graphics processors, memories, I/O, mixed signal modules, application specific hardware, Intellectual Properties (IP), peripherals, etc. We can now see small size appliances like satellite receivers, DVD systems, PlayStations, and others on a single chip.

1.3 System-on-Chip

The need for small and cheap electronic devices that perform many functions, and the ability to increase the number of transistors on one chip, forced a new product called the System-on-Chip (SoC) term to be emerged. SoC is a chip that does the function of a whole system by combining more than one processor and functional unit on one chip that can run more than one operating system. SoC can collect general purpose processors, DSP, graphics processors, memories, I/O, mixed signal modules, application specific hardware, Intellectual Properties (IP) cores, peripherals, etc on a single chip. SoC showed better performance and lower price than conventional designs.

1.3.1 Shared Bus

Along the past decades, shared bus was the conventional connecting scheme for on-chip modules. In shared bus, all modules are connected on one global bus. A bus is a dedicated number of wires mostly more than the bit width of the processor Arithmetic Logic Unit (ALU). Figure 1.1 shows an example of a typical shared bus connecting eight tiles.

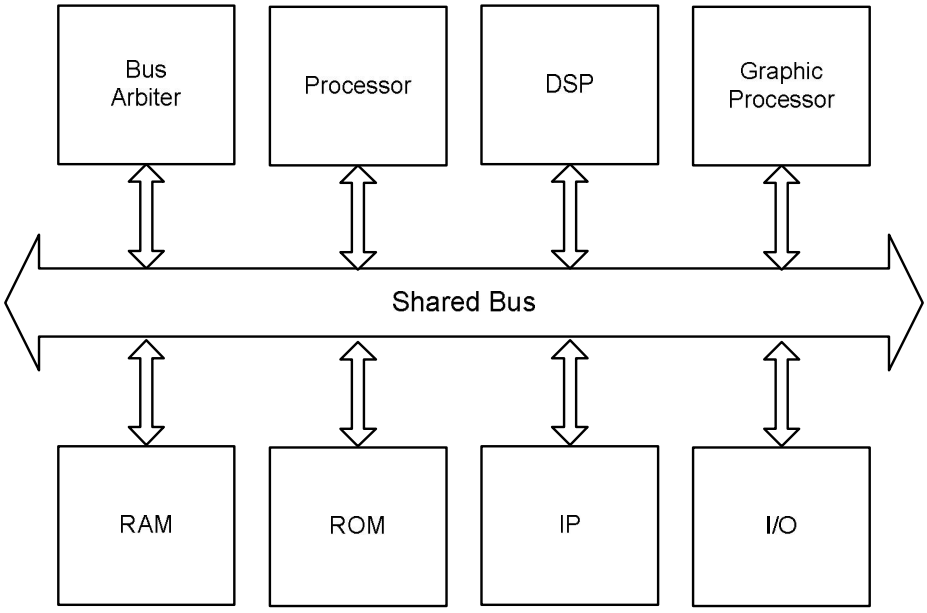


Figure 1.1: Example of a typical shared bus connecting eight tiles.

Shared bus communication scheme had a drawback that no more than one module can use the bus at a time. Hence, a centralized arbitration mechanism has to be well designed and inserted to handle the communication. The arbitration protocol requires the master to request to access the bus from the bus arbiter. The arbiter then grants one request at a time. One master gets a grant to access the bus whereas no other master uses it. So, contention can occur if more than one master is requesting the bus at the same time. The arbiter solves the contention according to the priority or a round robin manner. If round robin is used, requests are served one after another according the order of the requests. In priority arbitration, the master with lower priority has to wait until higher priority masters free the bus. The master sends the command to the slave and waits for the slave to respond. Unfortunately, other masters have to wait until the slave respond to the request and that wastes time for other masters to gain communication.

Split transaction mechanism is proposed to reduce the wasted time. In split transaction, masters free the bus after they send their commands to the slaves and slaves have to request the bus for themselves from the bus arbiter. Even

using split transaction, resource utilization is still small and system performance is low.

Additionally, when the number of modules on the bus increases, the arbitration process becomes very complex. Bus arbitration, which is needed to organize communication between the increasing numbers of modules in bus based SoC, becomes a bottleneck because of the contention happening between the increased modules for getting access to the shared bus. Employing priority levels to the bus member modules also increases the complexity. Arbitration complexity and contention between modules add more latency (delay) to the communication.

The need for higher performance in SoC required increasing the operating frequency. But as the size of SoC grows, the shared bus has to be very long (will turn to a global wire) and will be a limiting factor of achieving SoC's operation goals. Long distance will increase the load on the driving gates because the increased wire capacitance and resistance. That in turn will cause great drop in the data logic level voltage in addition to increase in the power consumption and heat dissipation. Higher power consumption required applying Dynamic Power Management (DPM), which means powering off unused modules [12][17] and therefore increasing the overhead in area by the added DPM control modules. To solve the drop in logic level we have to increase the operating voltage or insert repeaters and amplifiers to compensate the degradation in signal voltage. Long distance and inserting repeaters will increase parasitic capacitances. Higher capacitance and higher voltage will increase power consumption and decrease the maximum frequency in which the system can operate and this results in decreasing the performance. Reducing voltage swing to reduce power led to increased transmission errors on the global wires, which means the degradation of the data integrity (correctness of data transmission).

For long wire lengths, more than one tenth of the signal frequency component that data is transmitted on, the wire will behave as a transmission line and starts to emit electromagnetic radiation. So, data will behave as a wave which will travel only on the boundary of the outer surface of the wire. Hence, designers need to take into account the matching of the impedance between the transmitter and the receiver. Crosstalk is another unavoidable problem with the prolonged global wires.

So, shared bus communication scheme is the best way to design small systems because it is able to serve a limited number of modules. For larger systems, there transmission delay will be a bottleneck due to contention, and arbitration will be more difficult to design. That urged the researchers to look for other connection architectures that could overcome the shared bus problems. Researchers since the end of the 20^{th} century are trying to develop other connecting architecture based on computer networks to enhance performance, reduce latency, increase resource utilization, and serve the increasing number of modules in the SoC. This is realized by connecting the SoC using Networks-on-Chip.

1.3.2 Networks-on-Chip

As a solution to the limitations of shared bus, researchers proposed connecting the modules of SoC as networks. Small buses connecting small routers, each router connects one module in a mesh or similar network topologies is a new connection scheme called Networks-on-Chip (NoC).

Current NoC based architectures are still in their infancy and, only few companies (mainly not mainstream oriented) are starting to develop and to commercialize them in specific domains such as transaction intensive applications and network processing [18].

TILEPro64 is an example of a multi-core processor (Tile processor) manufactured by Tilera. It consists of a cache-coherent mesh network of 64 "tiles", where each tile houses a general purpose processor, cache, and a non-blocking router [19]. Tile Gx-8100 is the latest product of Tilera. It consists of 100 64-bit processor cores, where each core is connected by five terabit switches connected to five separate mesh networks [19].

Current manufacturing technologies using shared bus do not allow to integrate more than a dozen of cores of medium complexity while the NoC approach provides more performance and less power (over traditional bus-based solutions) when the number of interconnected units increase. However, in the near future, as predicted by many, CMOS technologies will continue to shrink and the number of cores integrated on a chip will increase. NoC based interconnects will start–hopefully–to reach mainstream products [18].

1.4 Goals

This thesis contributes to this field of research by proposing new topologies for the future many-core NoC. Then, presents three large interconnection router designs and compare their efficiency to the conventional small routers used in current NoC topologies.

1.5 Outline

This chapter sheds light on the advances in semiconductors, and then discusses the need for increasing the functionality of chips due to the compaction of electronic devices.

Chapter 2 discusses the details of NoC architectures and its connection schemes. It also sheds the light on power consumption in NoC. Then the related previous work done in NoC is discussed, especially in switch fabrics.

Routers building blocks and implementations are discussed at the end of the chapter.

Chapter 3 discusses the implementation of each router component in details.

Chapter 4 provides a detailed overview on the simulation environment designed to test the routers.

Chapter 5 discusses the performance evaluation results.

Chapter 6 provides conclusions and discusses work extensions.

Chapter 2

Networks-on-Chip

2.1 Introduction

Computer networking became a well-established field in the recent decades. It provided the ability to connect large number of computers with high bandwidth and high speed. Researchers began thinking of connecting cores like a computer network but on very small scale, the scale of semiconductor chip. Hence, the advent of the expression "Networks-on-Chip (NoC)". Figure 2.1 shows an example of connecting sixteen cores using a 2-D mesh topology NoC.

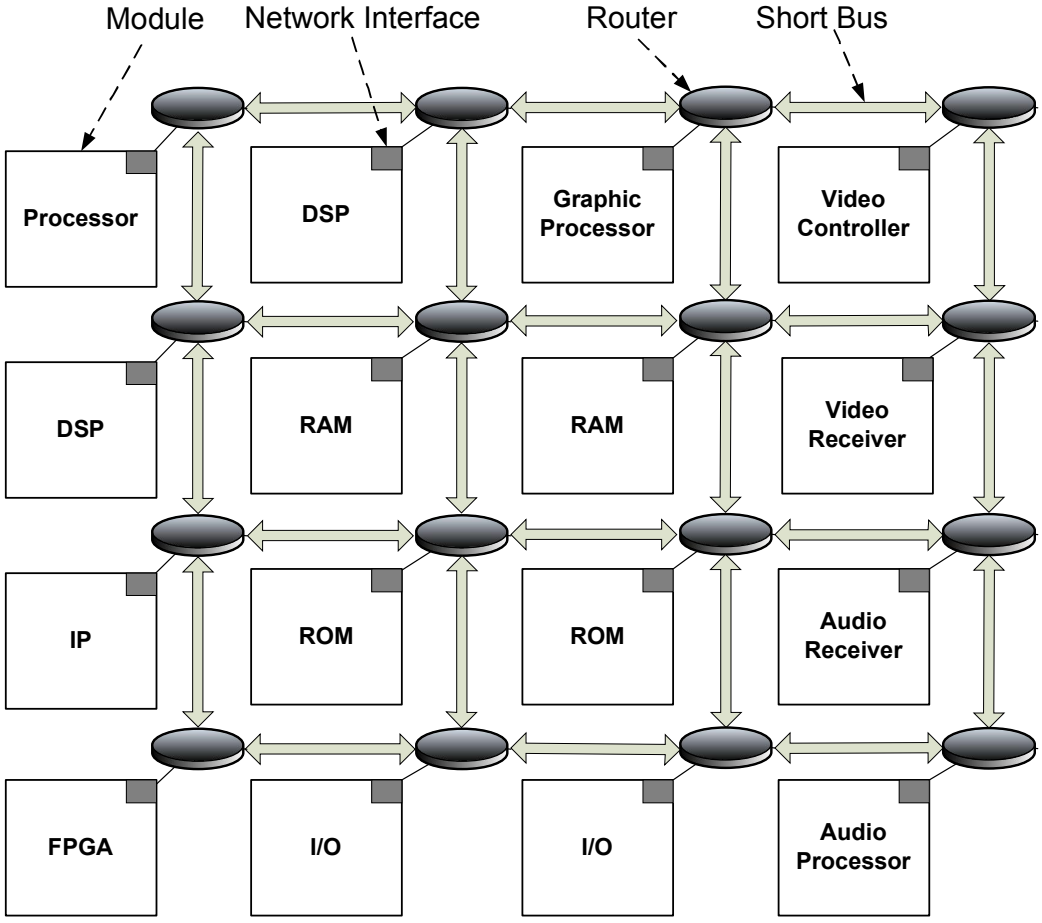


Figure 2.1: Sixteen cores connected as a 2D mesh topology NoC.

2.2 NoC Components

NoC replaces shared-bus connection scheme by a collection of switches (which is known in NoC as routers), network interfaces, and short distanced wires (short buses) where data is divided into small units and put into a special format (flits and packets) containing the address of the destination module. The data represented by flits and packets does not have to be delivered to all modules simultaneously as in shared-bus architecture, but it uses the destination addresses to go only from the source node to the destination. The following subsections discuss the main components used in NoC designs.

2.2.1 Modules

Modules are the components that contribute to the functions of the SoC. Modules could be master modules as processors. It could also be slave function units and memories. For only processing unit chips, if the chip is divided to fixed size tiles each tile contains the same size cores, same instruction set, same general clock, the NoC will be homogeneous. If the chip is divided in different size regions each region contains a different size core, different functionality, each have different clock, in this case NoC will be heterogeneous. Heterogeneous systems are complex in design efforts but are more efficient in performance, area, and power consumption.

2.2.2 Network Interface

Each module has to include a network interface (NI) unit. The function of the NI is to deal with the module as a bus in one side, and to deal with the network router as another router in the other side. So, it has to packetize and de-packetize data from the module. All NI's in the network will be

the same in case of homogeneous modules and will differ in case of heterogeneous modules.

2.2.3 Topology

Topology is the layout pattern of connecting modules inside the network. The selection of the best suited topology depends on the type of modules in the system; in case of homogeneous modules, mesh, torus, and similar topologies are best suit for the nature of same size tiles; while in case of heterogeneous modules, topologies like star, hierarchical-star and fat tree are more relevant to the nature of different size regions.

2.2.4 Routers

Routers are the basic backbone of the NoC interconnection communication. NoC routers must be simple in construction and fast in operation to meet the on-chip interconnection requirement of low latency and high throughput. Designers also must target the power and area constraints. Choosing the router architecture and routing algorithm depends on the used topology. In mesh topology, small 5×5 router with x-y routing algorithm is appropriate. Whereas a topology like star, hierarchical-star, or fat tree need larger routers and other routing algorithms. More information about routers are discussed in section 2.8 and the next chapter discusses the implementation details of the routers which is the mail research point of this work.

2.3 Basic Advantages of NoC

NoC design paradigm has many advantages over the previous paradigm of bus SoC interconnection. The following subsections discuss the basic advantages of

NoC design paradigm and how those advantages contribute to the electronic devices industry.

2.3.1 Scalability

Scalability is the ability to increase the number of modules on the SoC. In bus based SoC, it takes long time to add new function units to an existing design because designers had to design and test the system from scratch. Bus arbitration, data integrity, loads, and others have to be tested. In NoC, adding new modules or function units became easier. You only have to add a new router and connect this router to the network by small bus or even connect the module to an existing router depending on the network design. On the other hand, companies can buy a ready designed and tested functions, IPs from other third-party companies and commercial off-the shelf (COTS) modules from IP vendors then use them in their projects. That effectively reduces the strong pressure time-to-market (TTM) demand. So, scalability is the ability to increase the number of modules easily by just inserting a router and a network interface and we do not have to rebuild the design from scratch (placing the modules and routing the buses) to extend our design.

2.3.2 Reusability

The concept of reusability emerged with NoC, where in order to design a totally new product or a new application, designers may use many simple pre-designed modules to make larger systems. This concept can reduce time of design and test where designers do not have to think about new architectures or testing the system from scratch, and instead, they test the interconnection communication in the network. In other words, reusability means design once and use many. Reusability and scalability reduces the time of testing the design by just testing the traffic instead of testing all the system specially the low level modules.

2.3.3 Reduced Communication Delay

Contrary to the nature of communication in bus based SoC where only one communication channel can be established between two nodes at a time. Connecting a SoC as a network enabled the nodes to establish many successful communication channels every time slot. That in most cases reduces the communication latency and the total operation time.

2.3.4 Reduced Communication Power Consumption

In bus based SoC, every message has to be delivered to all nodes on the bus, and every node has to check whether the message is destined to its own or not. But in NoC, messages are routed inside the network only to its destination by the communication routers. That reduces the unwanted power consumption in transmitting the messages to unwanted destination and also the power consumption in checking the destination in each node. More information about power consumption in CMOS circuits is discussed in section 2.6 and NoC power consumption is discussed in section 2.7.

2.4 NoC Layers and Communication Requirements

Marrying networks and VLSI does not mean porting TCP/IP to silicon, the complexity and latency of TCP/IP will not match the needs of modules on silicon. On-chip communication has to be fast and simple. Many protocols are suggested [19]. Dally *et al* discuss a 4×4 tiles, each tile containing a module and router in a mesh topology, the network presents a simple reliable datagram interface to each tile, the packets consist of 256-bit data field with 32-bit header.

To satisfy performance and energy requirements of NoC, importing the seven layers of computer networks will not match the nature of silicon chips.

Computer network layers can be shrunk to be only three layers: physical layer, architecture and control layer, and software layer [15][12] described briefly at the following subsections.

2.4.1 Physical Layer

It is concerned by the electrical transmission (zeros and ones) between modules and routers. Power consumption can be controlled by choosing appropriate voltage logic swing and clock frequency. Using low-swing signals (by lowering the driver supply voltage) reduces power because dynamic power is related to the square of supply voltage. Lower swing can increase errors due to supply and external noise. A compromise between logic swing and bit error rate must be analyzed. Using dynamic power management by gating the main clock to the unused module can reduce power.

2.4.2 Architecture and Control layer

This layer contains data link layer, network layer, and transport layer. It does the packetization, adds a redundancy bits for data integrity (to assure data be transmitted correctly), checks error, and routing for outgoing data, and depacketization for ingoing data. It checks for transmission errors and makes necessary detection and correction.

2.4.3 Software Layer

It is the end firmware programs on the module doing a specific job. In video processing, Picture-In-Picture (PIP) is an example. Other examples are in the field of general processing, graphic processing, etc.

2.5 Many-Core NoC

Along with increasing the number of modules, another expression emerged. Many-Core NoC means a huge number of modules is connected as a network on one chip rather than traditional multi-core NoC. Many-Core NoC requires enhanced networking components like new router architectures. The increased number of modules forced the essential need for reusability and scalability. In future many-core NoCs, heterogeneity of modules are the dominant feature. Hence, uniform shaped topologies like 2-D mesh will not be suitable because of the different sizes. So, other topologies like star will suit this nature. Hierarchical-star and fat-tree topologies are the best candidate for scaling up the star topology in larger number of heterogeneous modules. Using hierarchical-star and fat-tree topologies with larger routers reduces the number of hops, hence reduces the latency and power consumption. Figure 2.2 shows a part of hierarchical-star topology employing large routers.

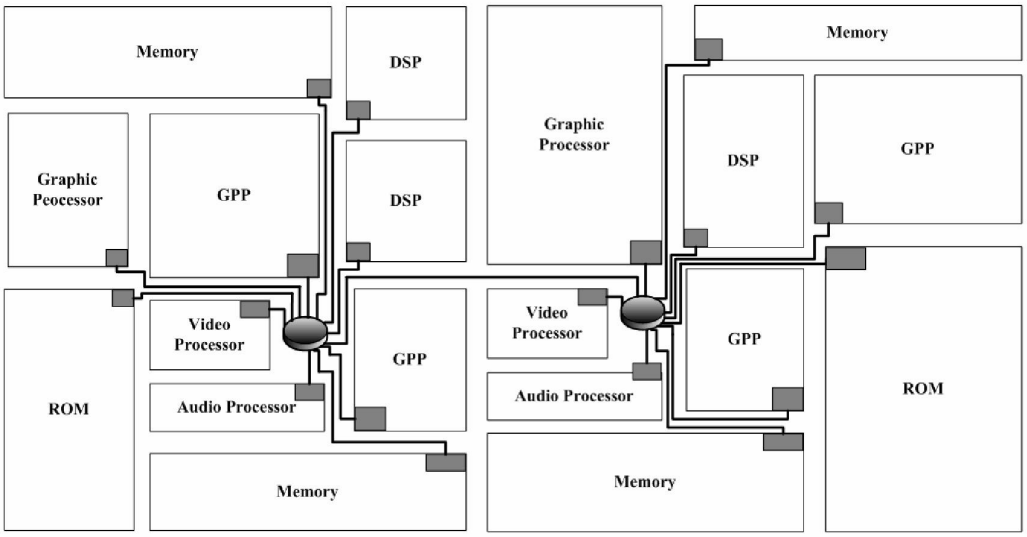


Figure 2.2: Part of a hierarchical-star topology employing large routers.

2.6 Power Consumption

A substantial challenge facing designers of high-performance computing processors is the need for significant reduction in energy and power consumption. The International Technology Roadmap for Semiconductors (ITRS) [13] highlights system power consumption as the limiting factor in developing systems below the 50-nm technology point. Moreover, interconnection networks dissipate a significant fraction of the total system power budget, which is expected to grow in the future [25].

2.6.1 Power versus Energy

Energy and power are commonly defined in terms of the work that a system performs. Energy is the total electrical energy consumed by device doing a certain job in a certain time, whereas power is the rate at which the system consumes electrical energy while performing the work.

$$P = \frac{w}{T} \ , \ E = P * T$$

Where P is power, E is energy, T is time interval, and W is the total work performed in that interval. Power and energy are two of the most important concerns in design of any SoC especially in battery-powered devices like laptops, PDA's, mobile equipments, etc. Techniques that reduce power do not necessarily reduce energy. For example, decreasing frequency will decrease power, but increase the time to finish a certain job. So, energy will be the same.

2.6.2 Power in CMOS Circuits

Power consumption in CMOS (Complementary Metal Oxide Semiconductor) silicon circuits has main sources: static power, dynamic power, and short-circuit power. Short-circuit power (due to the current passing for short time from supply to ground through NMOS and PMOS while switching) is

negligible compared to the other two power types. Dynamic power dissipation is the result of switching activity and is ideally the only mode of power dissipation in CMOS circuitry. Dynamic power dissipation is primarily due to charging of capacitative load associated with output wires and gates of subsequent transistors. As the following equation shows, dynamic power depends on four parameters, namely, a switching activity factor ($\propto$), physical capacitance (C), supply voltage (V), and the clock frequency (f) [33].

$$P_{dynamic} = 1/2 \propto fCV^2$$

Dynamic power has two sources: transistor (gate) capacitance and metal wire capacitance. As technology scales, these power sources scale differently [21] as in the following table:

Normalized power scaling factors.

	100 nm	70nm	50nm	35nm
Transistor Capacitance	1	0.78	0.65	0.66
Wire Capacitance	1	0.94	0.89	0.85
Static Power	1	2.01	3.35	4.30

Static (leakage) power (P_{ststic}) comes from leakage current from source to drain of the CMOS transistor. It is independent of the switching activity (logic 0 to 1 and 1 to 0). The previous table shows the effect of technology scaling on the static power. As transistors become smaller, static power increases dramatically. As the following equation illustrates, Static power is the product of the supply voltage (V) and leakage current (I_{leak}):

$$P_{static} = V \times I_{leak}$$

Gate level power simulators take long time to estimate the power consumption (days for typical trace). This is because these programs calculate the previous equations for each transistor. This time is valuable in designing commercial products especially if the design has to be changed many times to meet area or

power constraints. So, techniques for estimating power consumption is needed to quickly extract a fast power estimation model [31]

2.7 Power Consumption in NoC

NoC provide less power consumption with respect to shared bus SoC. The long shared bus is replaced by small busses between routers. So, data does not have to travel long distances to every module on the bus, but it finds its way to the destination module only. So, we save unwanted power consumed in non-intended modules by checking the data destination as in bus based systems. NoC also alleviates performance where we can increase easily the operating frequency since capacitance decreased due to short distances.

Power consumption in NoC systems is largely affected by on-chip networking (routers, network interfaces and interconnect wires), where the on-chip networking consumes about 36% of the overall system power consumption [22]. Power consumption in on-chip networking is affected by the following factors:

- Router architecture
- Topology
- Voltage swing on global links which connect clusters inside chip
- On-Chip serialization
- Flit (physical digit) size

2.7.1 Power Consumption in Routers

Routers contain different components; each contributes to the total power consumption. Router is comprised of the switching core, input units, arbitration unit, and the output unit. The following subsections discuss each component.

2.7.1.1 The Switching Core

Power consumption in router switching core comes from three different components [23]:

- The internal switch fabric
- The internal buffers
- The wires which connect the router components inside.

There are other components that consume low power like arbiter which can be neglected.

A. The Internal Switch Fabric

The switch fabric is used to connect the chosen output and input ports. The widely used switch fabric architectures are:

a. Crossbar

Crossbar has the benefit of being free of interconnect contention. However, the power consumption will be very high for switch fabrics with large port numbers because the number of crosspoints grows exponentially (N^2). The Crossbar is the most relevant for small switches. It is overly expensive for sizes above 32 or 64 [23].

For N×N Crossbar switches, there are N inputs that can be connected to N outputs via N^2 switching nodes called crosspoints. The crosspoint is simply a switch that can be opened or closed by an external control. The Crossbar switch is interconnection contention-free because every input/output connection has its dedicated path. Figure 2.3 shows an example of 4×4 Crossbar switching core. The figure shows 16 crosspoints, each crosspoint is controlled by a control NO/OFF signal (C*x*,*y*) where *x* represents the input port while *y* represents the output port.

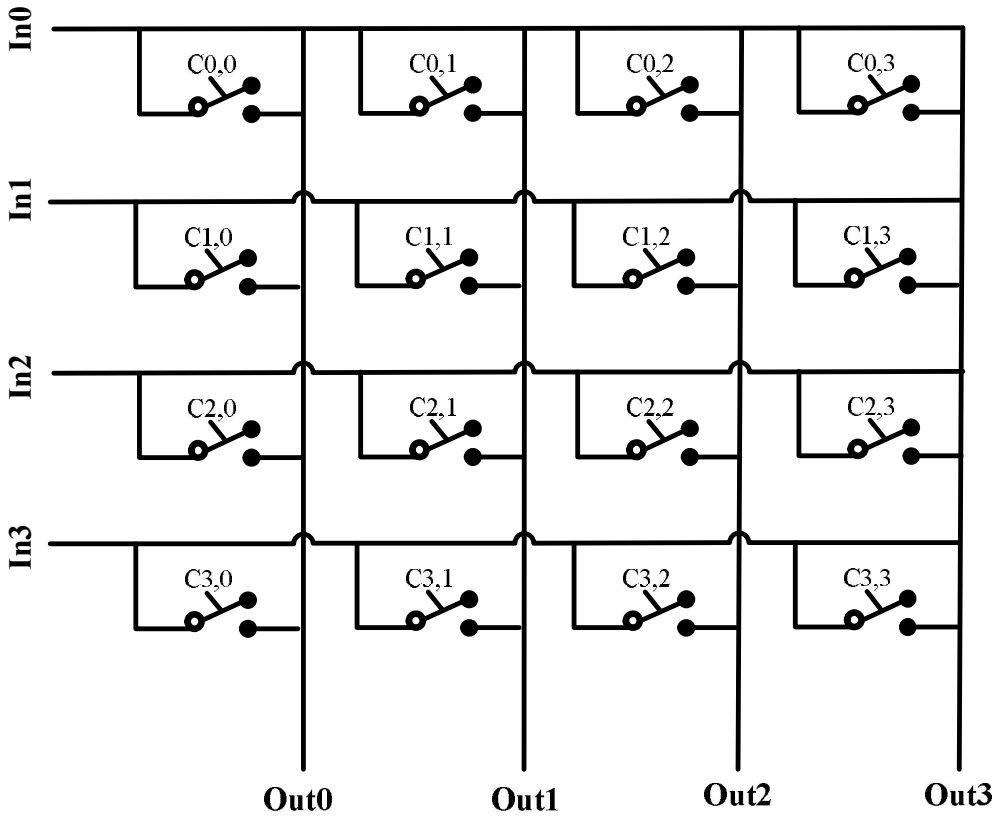


Figure 2.3: A 4×4 crossbar switching core.

b. Fully Connected

A fully-connected switch fabric uses MUXes to connect every input to the output. Each MUX is controlled by the arbiter that determines which input should be directed to the output. For N inputs fully-connected switching core, there are N multiplexers. Each multiplexer is M to one, where M is the number of inputs where all inputs are connected to the input of all the multiplexers and each multiplexer output is connected to the output.

The Fully-Connected Switching Core is interconnection contention-free because every input/output connection has its dedicated path through a dedicated multiplexer. The logic gate construction of a MUX enlarges exponentially with the number of inputs (N). Hence, its power consumption and area scale up exponentially with the number of inputs. Figure 2.4 shows an example of a 4×4 Fully-Connected switching core.

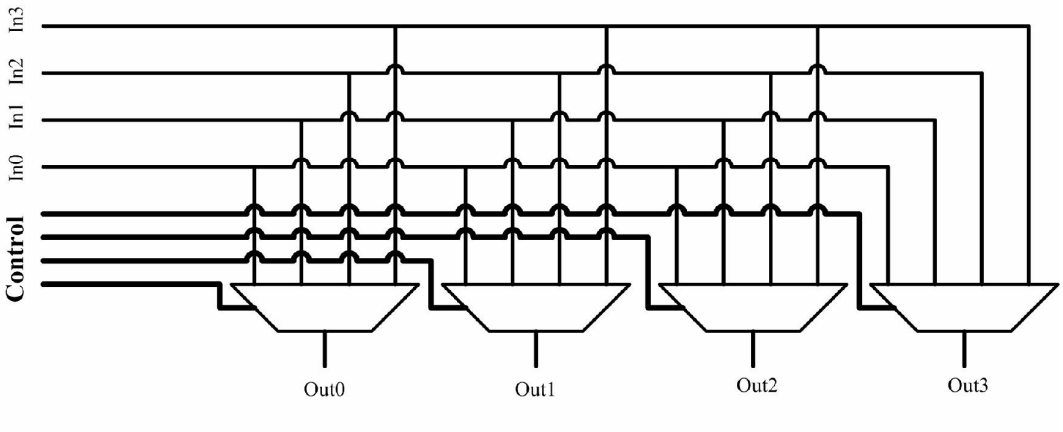


Figure 2.4: A 4×4 fully-connected switching core.

c. Banyan and Batcher-Banyan

Banyan switch is one of the multi-stage families of switching fabric architectures discussed in [41]. Banyan network is an isomorphic variation of Butterfly network topology where the switching nodes are connected as a multi-stage network.

The basic switching component of the Banyan switch is the switching node. The switching node is a complete self-routing 2×2 switch where it is able to route the input packet from any input port to any output port without any intervention from other control units. Figure 2.5 shows the block diagram of the switching node.

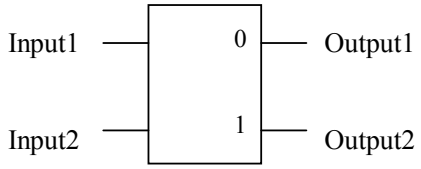


Figure 2.5: Block diagram of Banyan switching node.

Any larger switch is comprised of a collection of these small 2×2 nodes connected together in a multi stage interconnection network. The number of stages is determined by the number of input/output ports by the relation:

Number of stages = $\log_2 N$; where N is number of input/output ports.

Each stage contains ½N switching nodes. The switching nodes can be connected together according to one of four connection network classes as discussed in [42]:

1. Baseline network.
2. Narrow-sense banyan network, or
3. Reverse shuffle-exchange network,
4. Shuffle-exchange (omega) network,

Figure 2.6 shows the various connections of the Banyan network. The principal common properties of these networks are [42]:

1. There is exactly one path from any input to any output,
2. They have the self routing capability using the destination address,
3. The network is comprised of $\log_2 N$ stages, where N is the number of input/output ports and each stage contains N/2 switching nodes,
4. Very attractive to VLSI implementation.

The Banyan switch fabric is a self routing switch, it means that the packet finds its way to its destination by the switching nodes only with no intervention from the arbitration unit like other switching cores but all routings depend on the address in the header of the packet.

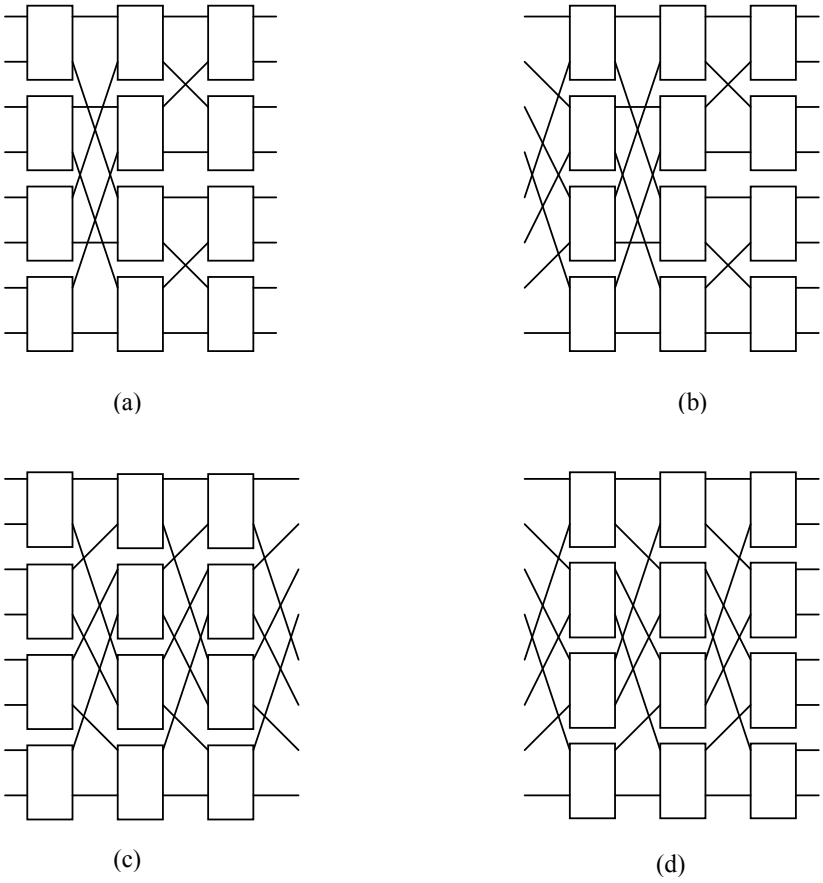


Figure 2.6: The four classes of Banyan connection network: (a) baseline network; (b) narrow-sense Banyan network; (c) reverse shuffle-exchange network; (d) shuffle-exchange (Omega) network.

The Banyan switch fabric is formed by a number of stages depending on the number of ports. For example, for eight input/output ports, there are three stages, each stage require one control bit to control the packet. So, there must be three bits needed as a destination address. The bits of destination address are used to route the packet from any input to any output, each bit is used for a different stage of the banyan connection network. The first bit of the destination address controls the switching node in the first stage, second bit controls the node of the second stage, and third bit controls the node in the third column. Figure 2.7 illustrates the mechanism of self routing in an 8×8 narrow-sense Banyan connection network.

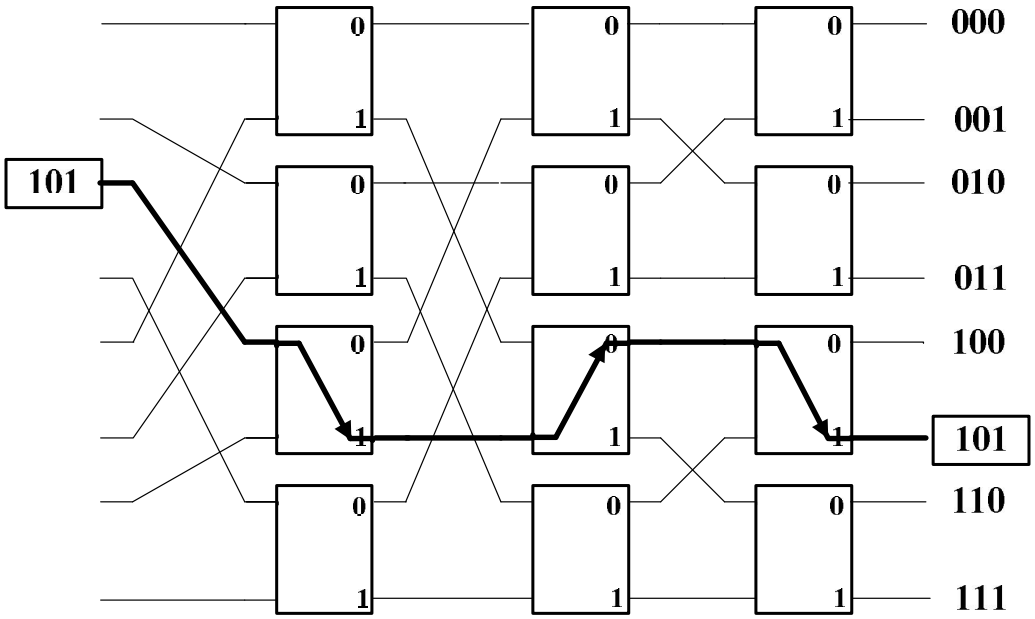


Figure 2.7: The mechanism of self routing.

The figure shows a packet is destined to port 101, the bold arrows show the routing paths. On the left hand side, the packet enters the third switching element in the first stage, the most significant bit is one causing the packet to be routed to the lower output port, and the address will be rotated left one turn. Then the packet enters the third switching element of the second column, the most significant bit became zero so, the packet is routed to the upper port, and the address will be rotated left one turn. The packet then enters the third switching node in the third column and out from the lower port because the most significant bit of destination address became one, and the address will be rotated left one turn to return to its original state.

Due to the fact this switch architecture utilizes the least switching nodes among all other switching core architectures, some sections of each input-output path inside the switching fabric may be shared between other paths. Hence, it suffers from the problem of internal blocking or resource contention where an internal resource will be shared by two packets and that causes a collision. Figure 2.8 shows an example of internal blocking in narrow sense Banyan connection

network. The figure shows two packets entering the Banyan connection network; one with destination address 011 entering at port one; and the other with destination address 010 entering at port three. Both packets will have to share the lower output of the first node in the second stage.

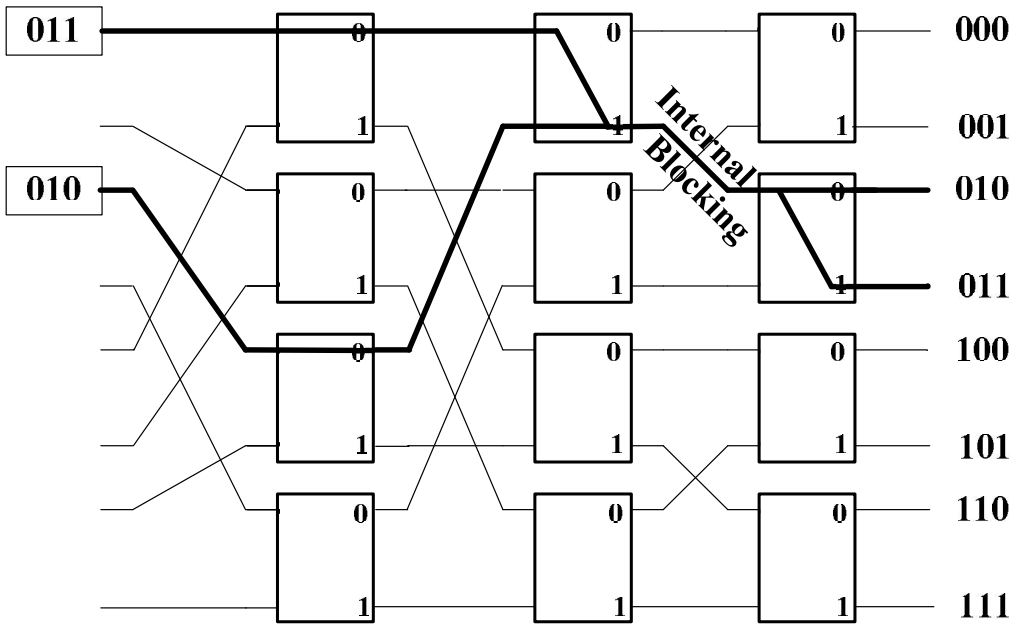


Figure 2.8: Internal blocking in an 8×8 Banyan connection network.

The internal blocking can be avoided if the following three conditions are met [42]:

1. There are no two input packets with the same destination address.
2. Input packets must be arranged in ascending or descending order according to their destination address.
3. There must not be any idle input port between any two active input ports (no gaps between arranged packets).

The first condition can be fulfilled by using an appropriate arbitration unit to guarantee that there will not be two packets with the same destination address reaching the switching core. But the second and third conditions can be fulfilled by using a sorting network that is able to arrange the input packets in ascending or descending order. Figure 2.9 shows the sorting network with respect to an 8×8 Banyan connection network.

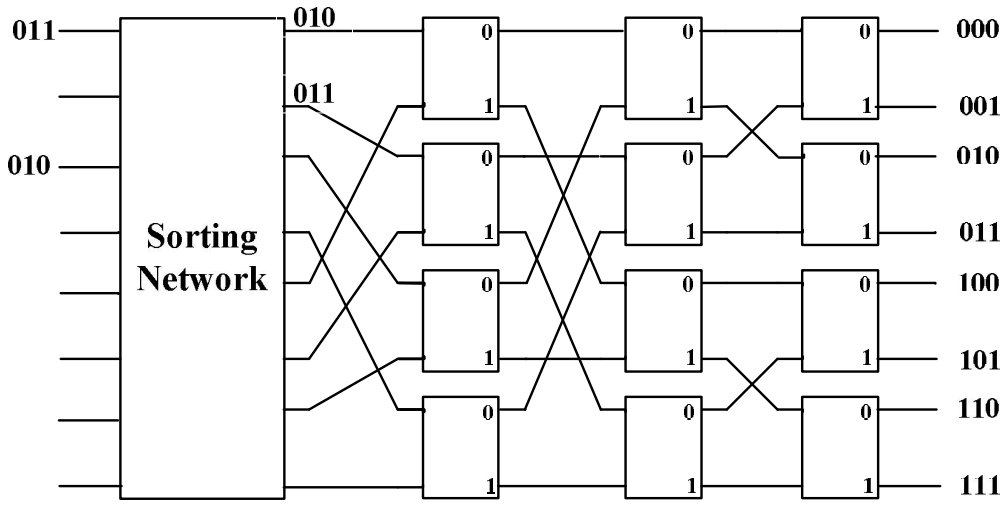


Figure 2.9: The sorting network with respect to an 8×8 Banyan connection network.

K. E. Batcher proposed a sorting network for the Banyan switch fabric [43]. The function of the sorting network is to concentrate the entering packets to the upper or lower ports of the Banyan interconnection network and arrange them in descending or ascending order according to their destination address. A Batcher sorting network consists of a collection of sorting nodes connected in stages with different sizes. The sorting node is comprised of a self-routing 2×2 sorting element. There are two types of the sorting elements used in Batcher network; the first can sort entering packets in ascending order whereas the other can sort entering packets in descending order. Figure 2.10 shows the block diagram of the two sorting elements.

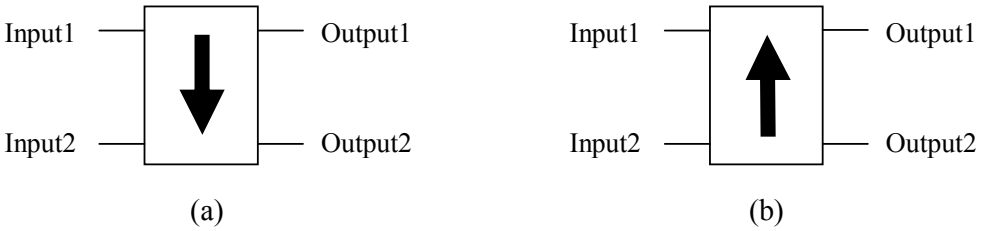


Figure 2.10: Block diagram of the two 2×2 sorting elements.
(a) descending element (b) ascending element.

Figure 2.10(a) shows the batcher_descend sorting element where if two packets entered, the packet with the smaller destination address always goes out at the upper output port (Output 1), if only one packet entered on any input, it will always go out at the upper output port (Output 1). Figure 2.10(b) shows the batcher_ascend sorting element where if two packets entered, the packet with the smaller destination address always goes out at the lower output port (Output 2), if only one packet entered on any input, it will always go out at the lower output port (Output 2).

A Batcher sorting network consists of $\log_2 N$ stages, where N is the number of input/output ports, and each stage contains different combination of both types of sorting elements in progression manner [42]. Figure 2.11 shows examples of 4×4 Batcher sorting networks consisting of two stages. The first stage is one column and the second stage is two columns. Figure 2.12 shows an example of 8×8 sorting network consisting of three stages. The first stage is one column, the second stage is two columns, and the third stage consists of three columns. Doubling the number of I/O leads to an additional stage, this stage has one column more than the previous stage.

We can observe that, if the order of the destinations of the first half input cells is ascending and that of the second half is descending, then the sorting network will sort the cells into an ascending list at the outputs. An 8×8 sorting network will be formed if an 8×8 merge network is preceded by two 4×4 merge networks and four 2×2 merge sorting elements. A completely random list of eight input cells will be first sorted into four sorted lists of two cells, then two sorted lists of four cells, and finally a sorted list of eight cells.

An $N \times N$ merge network consists of $\log_2 N$ stages and $(N \log_2 N)/2$ elements. A sorting network has $1+2+\ldots+\log_2 N = (\log_2 N)(\log_2 N+1)/2$ stages and $(N \log_2 N)(\log_2 N+1)/4$ elements.

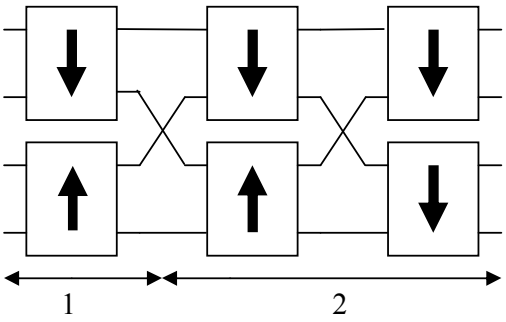


Figure 2.11: A 4×4 Batcher sorting network.

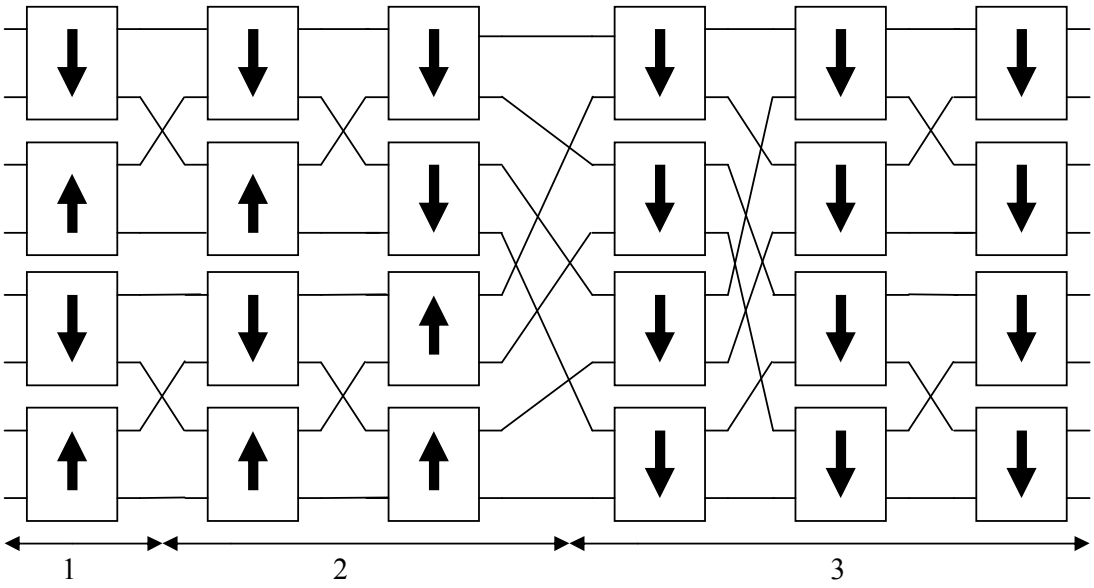


Figure 2.12: An 8×8 Batcher sorting network.

Batcher sorting network has a similar architecture to the construction of the Banyan network whose regularity and connection pattern are very amenable to VLSI implementation and self-routing property.

B. The Internal Buffers

The internal buffers are used to temporary store packets when contentions between packets occur. There are two types of contentions, destination contention and resource contention. Destination contention depends on the application executed by the system. Resource contention depends on the used switching fabric.

C. Interconnect Wires

For large routers (beyond 32×32), wire power consumption will dominate the overall power consumption of the router. Dynamic power consumption in wires happens only when there are changes in polarity of data, due to the wire capacitance of the interconnect [7].

2.7.1.2 Input Unit and Input Buffers

The input unit is the front stage of the router. There is an input unit for each input port. It handles address extraction, and it stores the incoming packet in the internal buffer. There are many buffering strategies to handle storing the incoming packets, each differ in complexity, performance, area, and power. For example First-in-First-Out (FIFO) input buffering strategy is more simple and less power consumer than Virtual-Output-Queuing (VOQ) input buffering strategy.

Buffers are the most area [19] and power consumers in NoC routers. They consume 64% of the total router leakage power [24], and share a great deal in router dynamic power [7][23][25]. Buffers can be implemented as either SRAMs or shift registers [21]. The shift register implementation is usually more expensive in terms of energy because it uses more transistors than the SRAM implementation. But, the shift register operations (read, write, shift) only involve occupied cells, while SRAM operations (read, write) involve all cells due to the global bitline and wordline wiring. So shift registers may consume less energy than SRAMs when the buffer utilization is below a certain threshold, and the opposite is true when the buffer utilization is above this threshold.

The shift register implementation is still viable at 100 nm technology with relatively smaller buffer size and lower buffer utilization, but is absolutely not an option at 35 nm technology. This is mainly caused by the rapidly increasing static power. So at 35nm technology, the advantage of fewer activities is

completely overwhelmed by the disadvantage of more transistors [25]. For a global wire in 180nm technology, the wire capacitance is around 0.5 fF/μm. Using these estimations, under 3.3V, storing a packet in buffer consumes far more energy than transmitting the packet on interconnect wires. This "buffer penalty" indicates that energy consumed in buffers is a significant part of total energy consumption of switch fabrics [7].

2.7.1.3 Arbitration Unit

An arbitration unit is usually used to resolve the destination contention problem. The destination contention occurs when two or more input packets are headed to the same output port in any switching core (except using multi-path or space-division switching where there are multiple interconnections between each input/output ports in addition to using output units, output units in this case require multi-port high speed memory which is highly expensive). If there are more than one packet headed to the same output port, the arbitration unit fairly schedules these packets one after another in successive arbitration cycles to eliminate the contention.

The power consumption of the arbitration unit depends on its complexity. Choosing an arbitration unit depends on the buffering strategy used in the input buffered unit.

2.7.1.4 Output Unit

The main role of the output unit is to buffer the output packets from the switching fabric in case the switch fabric can route more than one packet to the same output in the same time as in the space division multiplexed switch fabrics where there are more than one path from any input to any output that can be implemented by duplicating any switching core. In this case, the output unit has to have multiple port high speed memory which is expensive in cost, area and power consumption. Other case of using output units is in time

division multiplexed switches, where packets are switched from input to output on different time slots and on higher frequency. That also requires a high speed memory which is expensive in cost and power consumption. Although switches with output queuing have been shown high performance, their higher hardware complexity has restricted their popularity [39].

2.7.2 Power Consumption in Topology

Network topology is the layout of interconnection between the network nodes. It defines how nodes are connected to each other. Network topology determines the number of hops and the wire lengths involved in data transmission, both critically influence the power consumption per transmission [21].

2.7.3 Power Consumption due to Voltage Swing

The global link is a bus (group of wires) that connects different clusters on the chip. It is usually a few millimeters long in a large SoC and represents high capacitance load to its driver. So, global link consumes higher power than a local link. Applying low-swing signaling can reduce its energy consumption significantly. Low-swing signaling means transmitting data using lower voltage V_{swing} instead of V_{DD}. Lee *et al*. studied scanning V_{swing} from 0.25 to 1.1 V with 50-mV step when signaling rates are 400 Mb/s, 800 Mb/s, and 1.6 Gb/s on a 5.2-mm metal2 wire of 0.5-um width and 1.1-um space which has 330-fF parasitic and 100-fF coupling capacitance values in 180-nm process technology [26]. They found that energy per transmitted bit increases with the increase of V_{swing}, while the energy of amplifying it back in the receiver decreases. They found that the optimal swing voltages according to the energy and delay product are 0.45, 0.40, and 0.30 V at 400 Mb/s 800 Mb/s, and 1.6 Gb/s signal rates, respectively. Based on their measurement results, the low-voltage goes down to 0.27 V without transmission error. Due to the low-swing signaling, the

power dissipation on the global link is reduced to 1/3 of that on a full-swing repeated link and no repeaters are used on the wires to avoid area overhead [26].

2.7.4 Power Consumption of On-Chip Serialization (OCS)

The OCS means reducing the number of wires and input/output ports by transmitting data on many clock cycles. For example, OCS equals two means transmitting data on two cycles. Whereas OSC equals four means transmitting data on four cycles, and so on.

In NoC implementation, the OCS technique is used to reduce the number of interconnect wires hence, the switch size. Such reductions affect a network's overall area and energy consumption.

Power consumption of a link is determined by the wire capacitance and driving buffer size. When the OCS is applied, the number of wires of a link is reduced, so that the wires can be placed with larger space within the allowed routing area, which reduces coupling capacitance, and thus power consumption. On the contrary, the OCS increases operation frequency of the link, so the driver size must be increased to support the high-speed signaling.

Lee *et al.* studied the OCS in [27]. They found that power on link decreases until 4:1 serialization, and it increases for higher values of OCS, i.e. 8, which is due to high-frequency signaling overhead in the driving buffers. They found that both mesh and star areas are minimized when OCS is 4. Because of the queuing buffer overhead and high-frequency operation in the links, further serialization is not energy efficient.

2.7.5 Power Consumption in Flit Size

A packet is the smallest unit that contains routing and sequencing information. Packets contain one or more flow control digits or flits. A flit is the smallest unit on which flow control is performed. A flit in turn is composed of one or more physical transfer units or phits. A phit is W-bits, the size of physical communication media or data that can be transmitted in one system clock cycle (data per cycle) or the width of the data channel.

A packet may be further divided into flits. Flit is the basic unit of bandwidth and storage allocation used by most control methods. Flits carry no routing and sequencing information and thus must follow the same path in the network and remain in order.

Larger packet size will occupy the intermediate node switches for longer time, causing other packets to be re-routed to non-shortest data paths, which leads to more contention that increase the number of hops needed for those packets.

As packet size increases, energy consumption on the interconnect network will increase, hence, increases the energy dissipated on the network. On the other hand, larger packets decrease the energy consumption on cache and memory where it will decrease the cache miss rate.

Flit size is related to the NoC bandwidth and should be turned to application requirements. Flit size largely determines the role of network buffering resources. Switch size increases with increasing flit size.

NoC was configured with two different flit sizes, namely 21 and 38 bit simulated with 0.13um power characterized technology library was used for synthesis, floor planning and place-and-route. The system with 38 bit flit consumes more power than 21 bit flit [49].

2.8 Related Previous Work

Most of previous works are focused on small router architectures connecting mesh or torus network topologies. Most of them focused on small crossbar switch fabric architectures employed in mesh topology, while limited research was done on Banyan architectures or large router architectures.

A simple NoC architecture was proposed in [19], where Dally *et al.* discussed a 4×4 tiles, each tile contains a module and router in a mesh topology. The network presents a simple reliable datagram interface to each tile, where the packets consist of 256 bit data field with 32 bit header.

In [7][23], Ye et. al. estimated the power consumption of four types of switch fabric architectures (Crossbar, Fully Connected, Banyan, and Batcher-Banyan) by tracing the dynamic power consumption with bit-level accuracy, which is the power consumed by a bit traveling from input to output port inside the router. This work also studied the influence of CMOS technology, load capacity, and number of ports on power consumption. Authors use Thomson model [28] to determine wire length and hence get wire capacitance to estimate interconnect wire power consumption. They validated their model using Synopsys Power compiler simulator. To calculate the total power from their model using bit energy, they simulated the switching activities inside the switch fabric using S-function of Simulink and generated TCP/IP random packet traffic. They come out by some observations. We are interested in the following conclusion: *Banyan switch has the lowest power consumption under low traffic throughput, as the throughput increases, the power consumption increases exponentially (because it is dominated by the power consumption on internal buffers activity due to its interconnect contention (internal blocking)). In the 32×32 configuration, the Banyan had the lowest power consumption when the traffic throughput is less than 35%. In larger networks, Banyan will consume more power than other architectures.*

In [25], Wang *et al.* studied the power saving, performance, and area of three new switch fabric microarchitectures (segmented crossbar, cut through crossbar, and write through buffer) targeting energy reduction in router components. Authors modeled the previous switch fabrics architectures in the on-chip networks of two CMPs – the MIT Raw [29] and the UT Austin TRIPS [48] and obtained their power profile by using Orion [30] which is a power-performance simulator for interconnection networks. Simulations show that these mechanisms can achieve significant power savings of 44.9% for synthetic uniform random traffic, and 37.9% for the TRIPS traffic traces, as compared to a baseline network configuration (defining the baseline network as follows:
2-D torus topology, 128-bit flits, 5 flits per packet, 2 virtual channels per port, 16-flit input buffer per virtual channel, and dimension-ordered routing). Segmented crossbar achieved 25%, 33.3%, and 37.5% power saving with 2, 3, and 4 segments, respectively. More segments may increase crossbar delay, but they do not necessarily save more power either. Segmenting has no impact on area. Cut-through crossbar reduces crossbar energy by reducing input/output lines capacitance, consumes roughly 1/4 energy of a matrix crossbar and an upper bound of power saving is 75%. A cut-through crossbar clearly takes up less area than a matrix crossbar of the same scale. Write-through buffer only saves buffer read operations, at the expense of a separate bypass path. The upper bound of write-through buffer power saving can reach 60%. In reality, because on-chip networks tend to have small buffer size and also due to switch competition, write-through buffers achieve much less power savings than the upper bound. It has no negative impact on network performance. Write-through buffer only incurs marginal area overhead due to the additional registers and MUXes.

In [31], Jeremy *et. al.* presented a methodology for obtaining a fast and accurate energy macro-model for a synthesizable packet switched NoC routers. The NoC router is decomposed into components and a model is built for each component using semi-automatic system containing linear regression [32] to

characterize each router component. The model takes into account clock power and leakage power.

Authors consider the model for an input-queued router; the output link controller is considered as pass-through wires that consume no energy. The capacitance of wires founded using logic synthesis tools once the length of wires is known and applied to the macro-model.

Router can be decomposed into the following components:

- Input Link Controller (ILC): the major power consuming contributors are the FIFO memory and it's control logic.
- Crossbar Switch: implemented as two MUXes
- Arbiter: energy can decomposed into storage and computation of destination port.
- Route Unit: for both the X-Y routing algorithm and street sign routing algorithm are purely combinational and are determined based on the address presented by the input link controller.

Modelsim was used for all the VHDL simulations. Synopsys PrimePower was used to obtain the power estimates and the power waveform. Validation between the macro-model and pre-layout gate level estimates shows that for the randomly generated test patterns, the energy estimation error had a mean absolute error of less than 5%.

In [33], Penolazzi took as a reference a simple deflective (hot-potato) switch [34] in M×N mesh in a Nostrum NoC environment (a project concerning the development of an NoC in KTH). The author took the bit energy approach introduced in [23] as a reference to his study. The model has been validated against Synopsys Power Compiler simulations and is accurate within a few percent.

Passas *et al* [35] studied the implementation of a crossbar switching fabric connecting 128 tiles [35]. Area and power of the crossbar switching fabric and the proposed 128 tiles are deeply studied till the placing and routing using 90nm CMOS standard-cells. The crossbar is 32 bit width designed to deliver a

32 bit packet in one system clock (no serialization). Area cost of the 128×128 switching fabric (control not included) was 6% of the total tile area.

This Page Intentionally Left Blank

Chapter 3

Design and Implementation Details

Routers are the basic backbone of the NoC interconnection communication. NoC routers must be simple in construction and fast in operation to meet the on-chip interconnection requirement of low latency and high throughput. Designers also must target the power and area constraints. Choosing the router architecture and routing algorithm depends on the used topology. In mesh topology which is used with small number of modules, small 5×5 Crossbar router with x-y routing algorithm is the best appropriate architecture. Whereas topologies like star, hierarchical-star, and fat-tree connecting large number of modules especially heterogeneous modules needs larger routers and other routing algorithms for reducing the hop count, hence power consumption and delay.

3.1 High Level of Implementation

This work takes into account a general methodology of implementing the on-chip interconnection network components because there is no unique universal standard for NoC protocols. So, the design may be mapped easily to any NoC protocol or to any company project.

This work presents large router architectures targeting the future many-core NoC employed by star, hierarchical-star, or fat-tree network topologies. This work presents three 128×128 routers which can be scaled up as needed with slight modifications.

All router designs in this work are based on synchronous transmission where the same clock frequency with the same phase is applied to all modules. The

first physical digit or phit (data per cycle or the width of data packet [36]) of ingress packets are detected using a separate signal called start of frame (*sf*). Upon receiving high level on the start of frame, the first phit (which contains header information) is sampled. At the second clock cycle after the high level of the start of frame, the second phit is sampled, and so on. For asynchronous systems, *sf* can be eliminated easily and a synchronization circuit must be attached to the input of the router. Figure 3.1 shows the start of frame with respect to the data and clock.

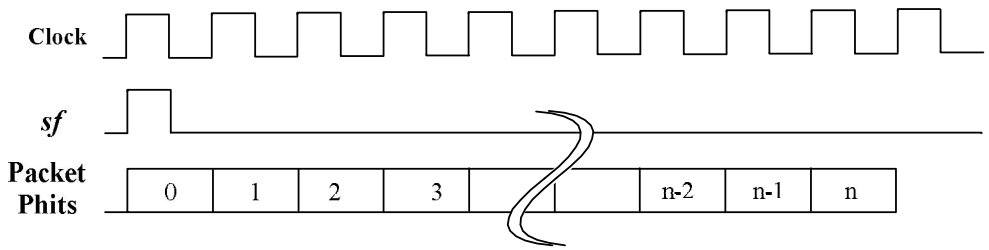


Figure **3**.1: Synchronous transmission of packets.

This work considers eight bits phit size or eight interconnecting wires between routers or between routers and network interface of modules. This is a moderate phit size where serialization ratio may be 4, 8, 16 or more for 32, 64, 128, or more for various packet sizes. The first phit of the packet carries the header of the packet. The header of the packet must hold the destination address where it is checked immediately and the buffering process is done according to it. The header can be increased to contain other useful information such as source address and type of packet, but destination address must be maintained in the least significant bits of the first phit. For other phit sizes, switch can be changed in the generalized code with no impact on the functionality as long as the router size (number of input/output) is considered. For example as in the designs in this thesis, 128×128 router needs at least seven bits phit width to contain the seven bits destination address.

Variable packet size has a less utilization of buffers and difficulties in manipulation, so this work assumes fixed packet size for simplicity. 32 bytes packet is considered moderate packet size that can be used for carrying data or commands and acknowledges. Packet size can be changed easily in the written generalized code with slight changes with no impact on the functionality.

3.2 Router Building Blocks

Any switch must contain four basic components; input unit, arbitration unit, switching core, and optional output unit. The general block diagram of switch units is illustrated in Figure 3.2.

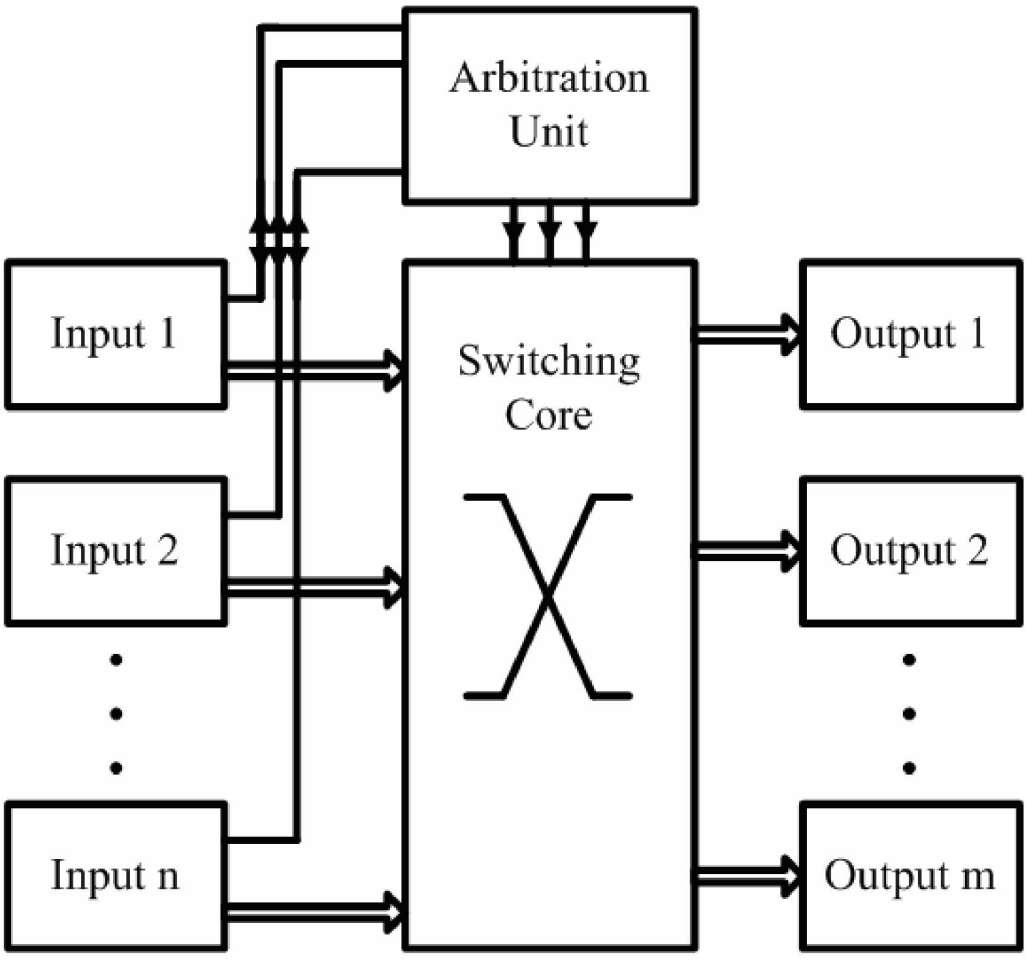


Figure 3.2: The main building blocks of an NoC Router.

In this work, three different router architectures are implemented. The first is First Input First Output (FIFO) input buffer with Batcher-Banyan, the second is Virtual Output Queuing (VOQ) input buffer with Batcher-Banyan and the last is VOQ with Crossbar. Each design is 128×128 input/output size. The designs are based on synchronous transmission where each input is synchronized and detected using separate one bit signal indicating the start of packet when pulsed high. The input port data width is eight bits wide (seven bits containing the destination address of 128 ports) which represent one phit (phit is the data transmitted in one clock cycle). The designs are based on fixed packet size. The packet can be transmitted in one clock cycle as one wide phit or divided on multiple cycles as small phit to reduce the number of wires and hence area.

All components of the switch architectures are developed and written in structural Register Transfer Level VHDL (VHSIC Hardware Description Language), where VHSIC stands for Very High Speed Integrated Circuits. VHDL is a hardware description language used in electronics field to describe hardware digital and mixed signal systems such as Application-Specific Integrated Circuits (ASIC) and Field-Programmable Gate Arrays (FPGA).

Each component is tested on Mentor Graphics ModelSim 6.5. The overall switches are then tested as a unit. A simulation environment resembling normal operations to test performance aspects, throughput and average packet delay under various traffic loads are developed.

Each router design is synthesized using Synopsys Design Compiler using TSMC 65 nm standard cell library to get the area and dynamic power, and then the designs are evaluated by calculating the efficiency of the routers. Two router efficiencies are defined, the first is the area efficiency which is throughput normalized to one area unit and the other is the power efficiency which is the throughput normalized to one dynamic power unit.

The following sections discuss in details the hardware implementation of each component of the router architectures.

3.3 Input Unit

The input unit is the front stage of the router. There is an input unit for each input port. It handles address extraction, and it stores the incoming packet in the internal buffer. There are many buffering strategies to handle storing the incoming packets, each differ in complexity, cost and performance.

In this work, two different input units are implemented; the first uses FIFO input buffering strategy, and the others use VOQ input buffering strategy. Both the two input units are discussed in the following subsections.

3.3.1 First-In-First-Out (FIFO) Input

FIFO input queuing buffering strategy is the most direct and simplest way of implementation where incoming packets are stored in one queue. The first input packet is stored at the head of the queue and the second is stored at the second place until the last packet which is stored at the tail of the queue. Any new coming packet is stored at the tail of the buffer. Figure 3.3 shows an example of 4×4 switch using the FIFO input buffering strategy.

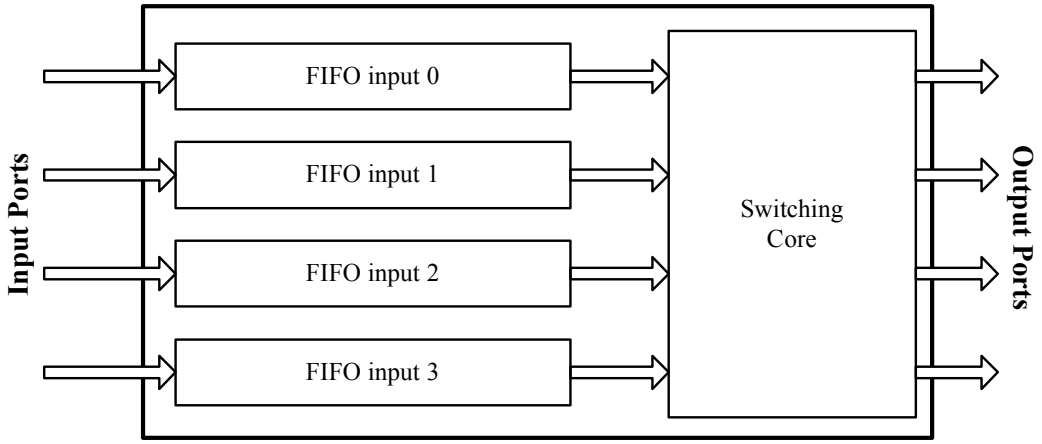


Figure **3**.3: A 4×4 switch using the FIFO input buffering strategy.

The drawback of this strategy is that if the head packet had to wait because its destination port is busy receiving from another input unit, other packets in the queue have to wait until the head packet could be served even if its destination port is idle. This problem is named Head of Line (HOL) blocking and limits the throughput to 58.6% [37].

The FIFO input unit stores the incoming packet in one queue buffer. It controls the buffering process of incoming packets, extracting the destination address, sending a request to the arbitration circuit, receiving the grant and sending the packet to the switch fabric. Figure 3.4 shows the high level schematic of the input unit.

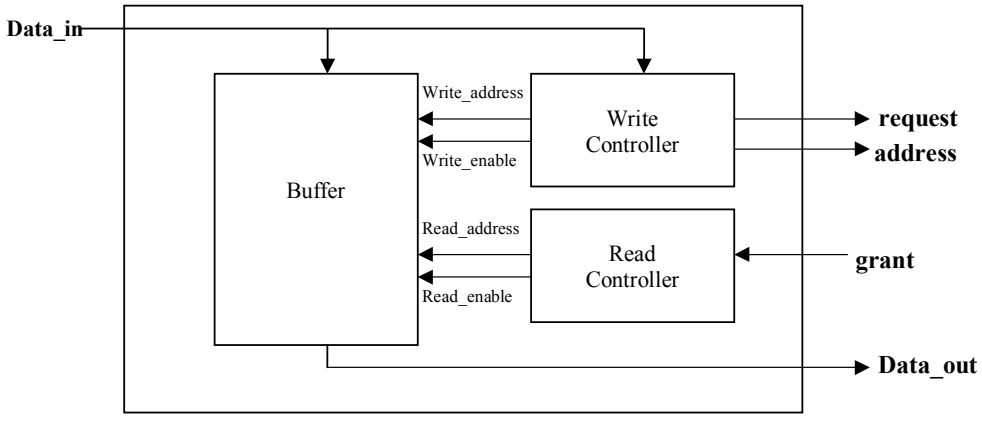


Figure **3**.4: The high level schematic of the input unit.

First, when a packet arrives to the input unit it is saved to the buffer. While buffering, the write controller extracts the destination address from the packet and sends it along with a request signal to the arbitration unit. After saving (queuing) the packet to the buffer, the input unit waits for a grant signal from the arbitration unit to start reading (dequeuing) the packet from the buffer. The input unit state machine uses a number of registers to control the buffer for queuing and dequeuing processes of the incoming packets. The following is a discussion of used signals and control registers. Signals and control registers are illustrated in the detailed schematic in Figure 3.9.

Empty flag:
It is a bit register memory named "empty" set to one when there are no packets in the buffer and reset to zero during writing of the first byte of the first packet.

Request:
The request is an output signal sets to one when the first byte of the first packet is written to the buffer and reset to zero when last packet is read from the buffer. It is simply the inverse of the empty flag.

Head:
A register holds the address of the first packet stored in the buffer.

Tail:
A register holds the address of the last packet stored in the buffer.

Write_counter:
A counter register used for counting the packet phits in write controller.

Read_counter:
A counter register used for counting the packet bytes in read controller.

Write_address:
A register holds the address of the current writing byte in memory.

Read_address:
A register holds the address of the current reading byte from memory.

Address_signal:
A table contains the destination addresses of all the packets in the buffer.

The following subsections describe in details the main components of the input unit.

3.3.1.1 The Buffer Memory

The input buffer consists of synchronous dual port memory. The dual port memory consists of an array of signal registers (D-flip flops) with one port for reading and another port for writing. There are two address lines in the design; the first is the write address (Wr_address) which determines where the data should be written, the other address is the read address (Rd_Address) which determines where the outgoing data are read from. There are separate enabling signals for writing (wr_en) and for reading (rd_en). The write and read operations is synchronized with the rising edge of the main clock. The dual port memory can manipulate writing incoming packet and reading outgoing packet on the same time on a separate data ports (data_in) for writing and (data_out) for reading.

The buffer is divided into banks. Each bank can hold one complete packet. The choice for buffer_width is a trade off between loss rate in a side and speed and router area in another side. The larger the buffer, the smaller the probability of buffer overflow and dropping packet, but will increase area on silicon , power consumption and time to queue and dequeue packets. Buffer memory code is general and can be easily changed as required. Figure 3.5 shows the schematic of the buffer memory.

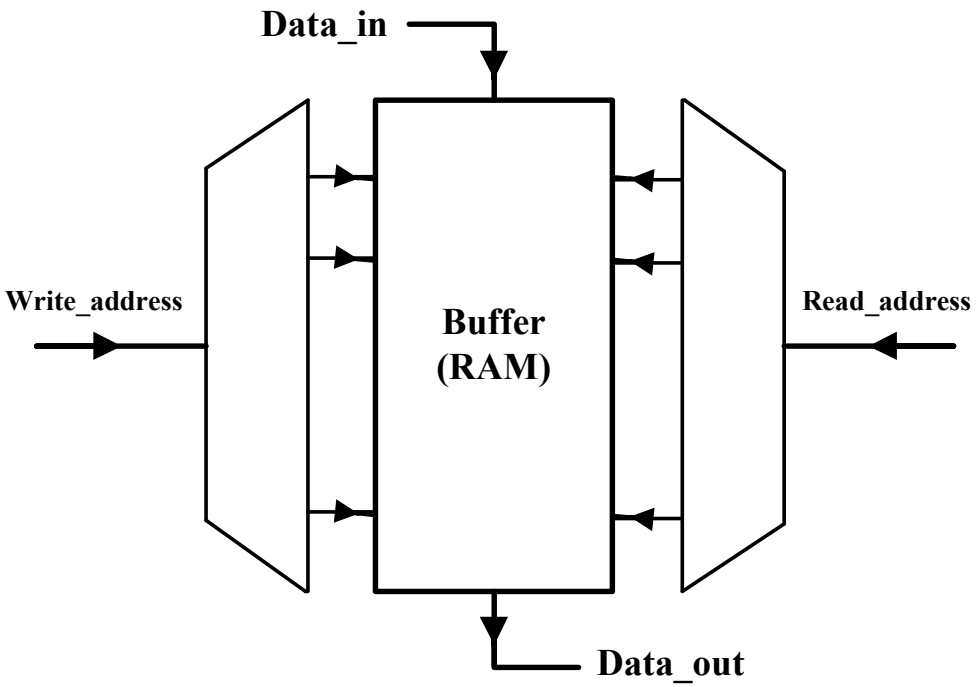


Figure **3**.5: Schematic of the buffer unit

3.3.1.2 Write Controller (Writing_sm)

The write controller consists of a state machine called *Writing_sm*. This state machine has three states operating at the rising edge of the main clock of the switch. All state transitions happen by a separate process, called *Wr_state_change*.

The condition of the reset is checked at the start of the *Writing_sm*. If active, the state machine does three actions; resetting the tail register to zero, disabling the buffer write enable, and resetting the state to state zero.

The state machine stays at state zero until the input start of frame (*sf_in*) of the incoming packet, (*sf_in* is sampled at the rising edge of the main clock of the switch). Upon detecting of *sf_in*, the circuits check whether the buffer if full, the buffer will be full if the *empty* flag is at logic one while the head and tail registers are equal. If the buffer is full the circuit will simply drop the packet. If

the buffer is not full, the writing controller calculates the writing address by appending (combining) both the tail register as highest significant bits and the Wr_counter as least significant bits.

After the calculation of the writing address, the buffer enabling signal (*Wr_en*) is asserted and the first byte of the incoming packet is written to the buffer with the calculated address. Writing counter (*Wr_counter*) is incremented each phit (clock cycle) until the end of the packet.

The write controller extracts the destination address from the first byte of the incoming packet and saves it in the address pointer table. Then, the state machine goes to state one.

At state one, the second byte of the packet is saved to the buffer, and the tail register is updated by incrementing it by one. Then the state machine goes to state two.

At state two, the rest of the packet is saved to the buffer. The write controller continues to save the packet until the writing counter reaches the end of the packet. It then stores the last byte from the packet then stops the writing counter. Then, the state machine goes to state zero. Figure 3.6 shows the algorithmic state machine of the writing controller.

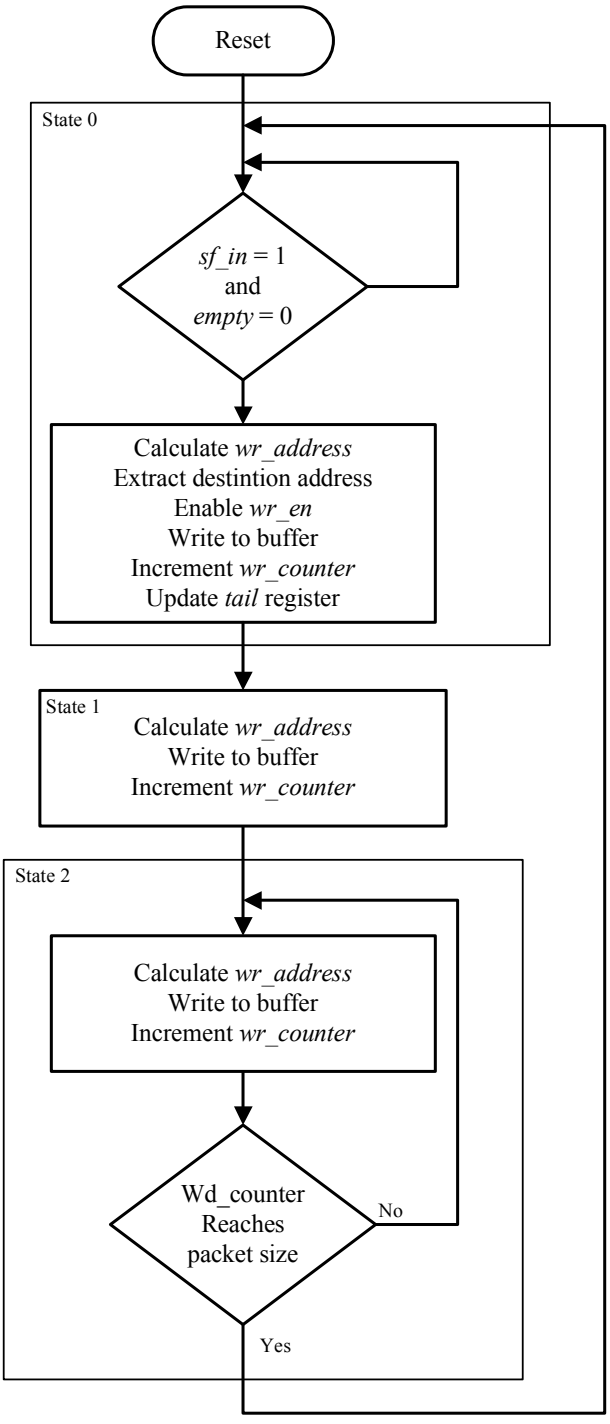


Figure **3**.6: Algorithmic state machine of the FIFO write controller.

3.3.1.3 Read Controller (Reading_sm)

The read controller consists of a state machine called *Reading_sm*. This state machine has three states that operate at the rising edge of the main clock of the switch. All state transitions happen by a separate process called *Rd_state_change*.

The condition of the reset is checked at the start of the *Reading_sm*. If reset is active, three actions are done; resetting the head register to zero, disabling the buffer read enable, and resetting the state to state zero.

The state machine stays at state zero until a grant signal arrives from the arbitration unit to allow the granted packet; which is the packet at the head of the buffer; to cross to the switching core (the grant signal is sampled at the rising edge of the main clock of the switch). Upon detecting the grant signal, the read controller calculates the reading address by appending (combining) both the head register as highest significant bits and the *Rd_counter* as least significant bits.

After the reading address is calculated, the buffer read enabling signal (*Rd_en*) is asserted and the first byte of the dequeued packet is read from buffer and put on the bus to the switching core at the rising edge of the clock by the buffer state machine. The head register is updated by decreasing one from it at the first state. Then the state machine goes to state one.

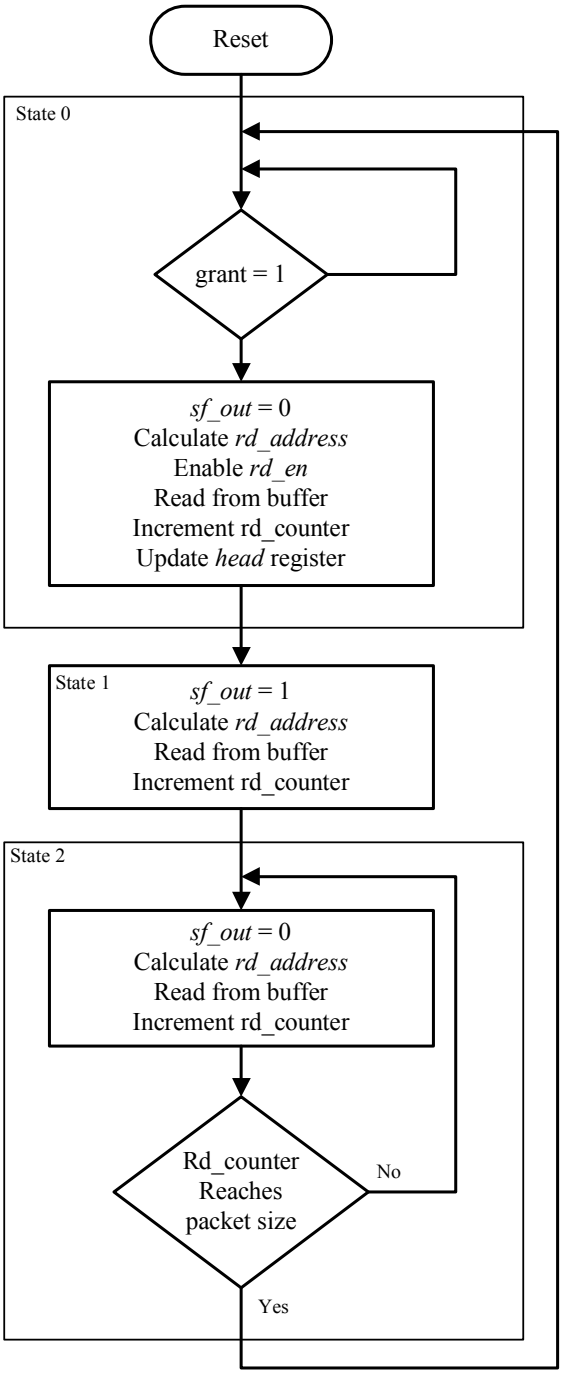


Figure **3**.7: Algorithmic state machine of the FIFO read controller.

At state one, the output start of packet (sf_out) is asserted to wait until the buffer memory to output the first phit of data. The next address is calculated. Then the state machine goes to state two.

At state two, the rest of the packet is read and sent to the switching core. The read controller continues to read the packet until the reading counter reaches the end of the packet then stops the writing counter. Then the state machine goes to state zero where the state machine disables the read enable of the buffer (reset Rd_en). Figure **3**.7 shows the algorithmic state machine of the read controller.

3.3.1.4 Buffer State Update (empty_sm)

This state machine operates at the rising edge of the clock to update the *empty* flag because both the writing controller and reading controller can not alter the state of any register. So, there must be a separate state machine to update the *empty* flag upon writing and reading from the buffer.

When reset is asserted, the circuit sets the *empty* flag to one to indicate empty memory and that ignores any previous packets in the buffer. At normal operation, it checks the state of the writing and reading state machines. If *empty* was set (the buffer is empty) and the next state of writing controller is one, it resets the *empty* flag to zero indicating a non-empty buffer. Else if the *head* register is equal to the *tail* register (indicating the last packet in the buffer) and if the next state of reading is one and the next state of writing was not one it sets the *empty* flag to one. The flow chart of the buffer state update is illustrated in Figure 3.8.

The overall schematic of the input unit is illustrated in Figure 3.9.

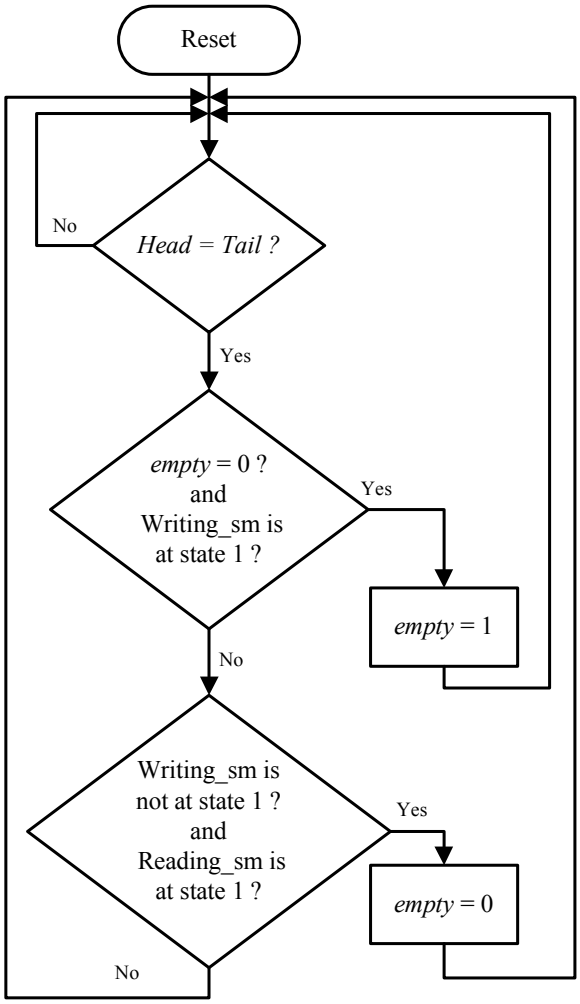


Figure **3**.8: Flow chart of the buffer state update state machine.

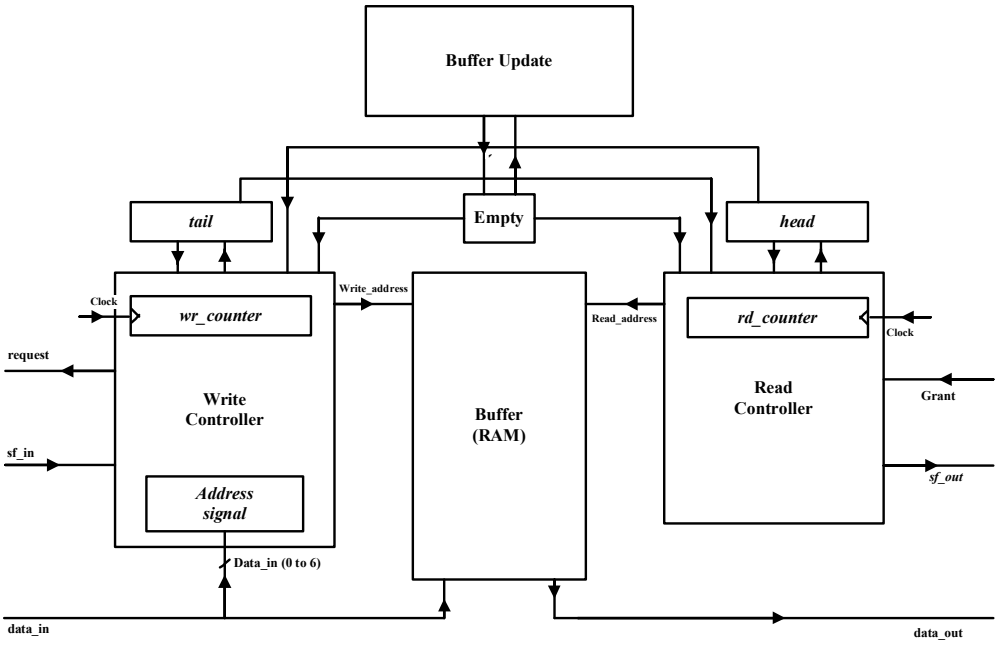


Figure **3**.9: Detailed schematic of the input unit.

3.3.2 Virtual-Output-Queuing (VOQ) Input

The degraded performance of the FIFO input queuing which limits the throughput to 58.6% [37] due to the Head of Line (HOL) directed the interest to a new input buffering strategy which enhanced the throughput and hence performance. The new buffering strategy is Virtual Output Queuing (VOQ). Modifying the FIFO input queuing to be VOQ enhanced the throughput to about 100% and also performance of the buffering unit [38].

VOQ buffering strategy divides the buffering area to a number of buffer queues equal to the number of output ports. Each queue holds packets with same destination address. Any new packet is stored at the tail of its dedicated queue according to its destination address. Virtual output queuing overcomes the HOL problem in the FIFO input queuing where each queue can issue a separate request and can be served independently. Figure 3.10 shows an example of a 4×4 switch using the VOQ input buffering strategy.

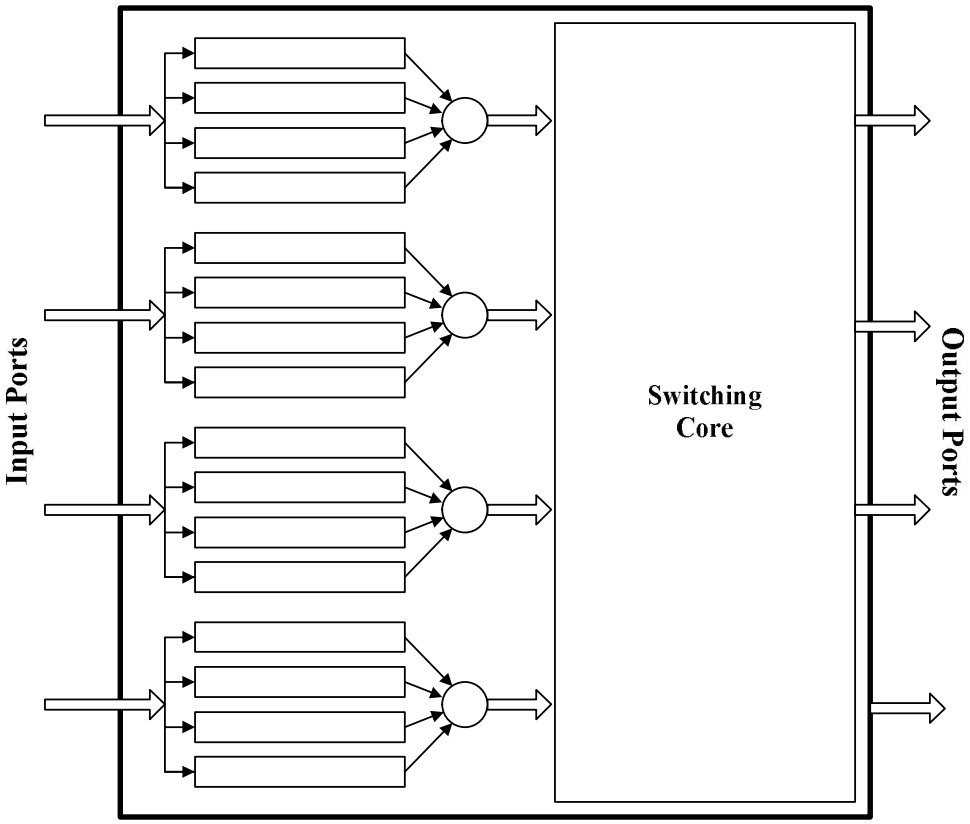


Figure **3**.10: Example of a 4×4 switch using the VOQ input buffering strategy.

3.3.2.1 Static vs. Dynamic Allocation VOQ

In static allocation VOQ, the virtual queues are assigned fixed sizes in the design phase. The traffic pattern dictates each queue share from the whole buffer. If the traffic is always directed to some outputs, the size of their dedicated virtual queues may be larger than the others. If the traffic is uniform or unknown, the buffer must be divided equally among the virtual queues. In most cases, the buffer size of each input port is divided equally between the virtual output queues. If any of the virtual output queues became full, it will not be able to receive a new packet and the packet will be dropped even if there are spaces in other queues. This is still one important disadvantage of static VOQ. The buffer for one small queue can be full while all the other small queues of the same input port can be empty.

Dynamic allocation VOQ means that the size of each virtual output queue is not fixed. The buffer area of the input port is kept one unit and dynamically divided in the run phase into a group of out queues as needed by the nature of the traffic. The use of dynamic buffer allocation results in better utilization of the buffer space, compared to the existing virtual channel switches with fixed size channels [39].

Dynamic allocation can decrease the number of dropped packets by approximately 30% on average, while also providing for increased reliability. In addition, dynamic allocation of buffers reduces packet loss. Reducing packet loss in queuing systems as the result of utilizing an effective buffer management system is an important issue in the design of switches. Reducing packet loss is equivalent to improving efficiency [40].

3.3.2.2 Linked-List Buffer Management Scheme

There are many management schemes to dynamic allocation control buffer memory like linked list, self compacting, and circular buffer [38]. For more information about other management schemes see [40].

The linked list buffer management scheme keeps a list (queue descriptor) for each virtual output queue and another list for additional queue called the free space queue which holds the rest and unoccupied memory blocks. Each queue list (descriptor) contains a flag that shows whether the block is empty or not, and two pointers; one points to the head of the queue and the other points to the tail of the queue. For each block, a pointer is needed to point to the next block that belongs to the same queue. Figure 3.11 shows an example of buffer and its related linked list registers. The figure shows a simple buffer comprised of four blocks (a block is a space for containing one packet) and two virtual output queues. For each queue, there is a queue descriptor which is comprised of *head* and *tail* registers which hold the first and last blocks, respectively, in the queue,

and *empty* flag shows whether the virtual queue is empty or not. The free space queue contains one remaining block at address 3. Queue 0 contains two blocks; the head block is at address 0 and the tail block at address 2. Queue 1 contains only one block at address 1. Each block has a *next* register which indicates where is the next block in the same queue.

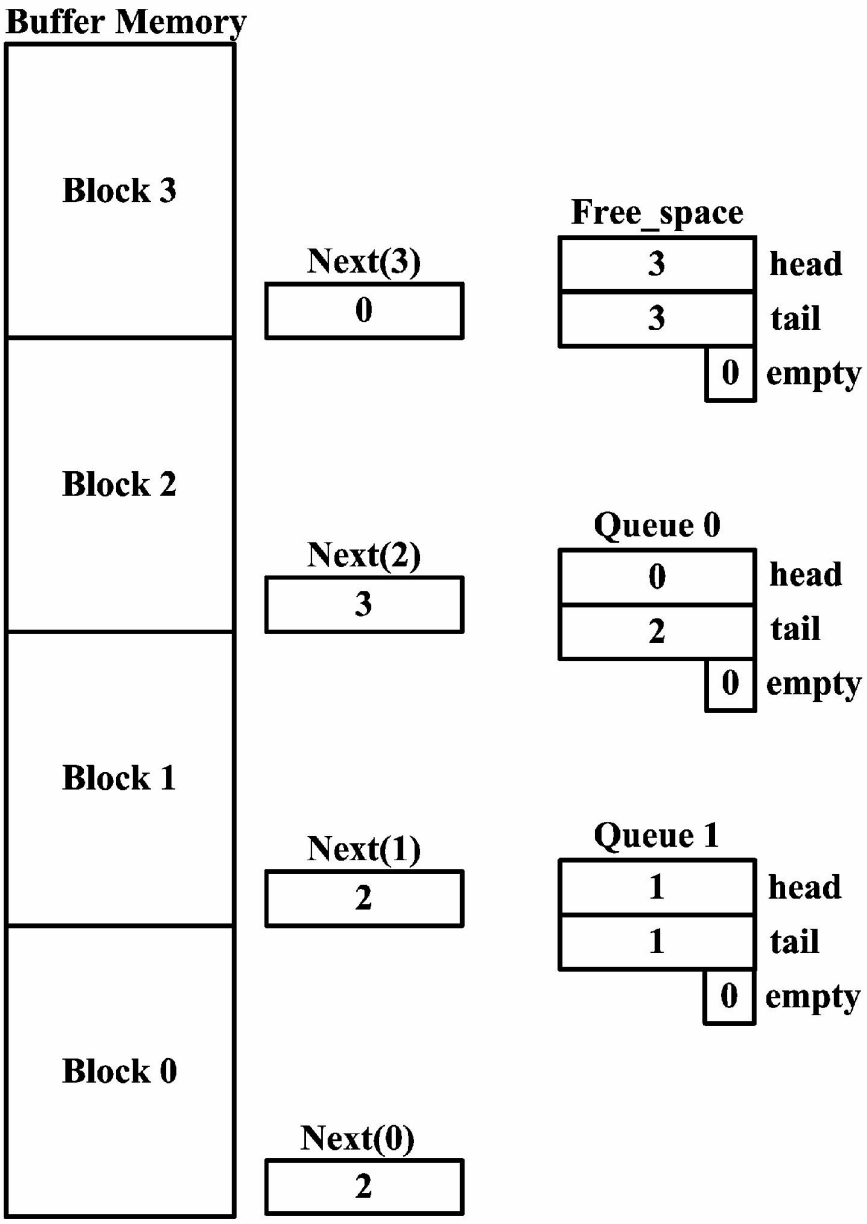


Figure **3**.11: Example of linked-list registers.

3.3.2.3 High Level Schematic

The input unit consists of the buffer memory in addition to three main sequential blocks *(Read controller, Write controller,* and *Linked_List Update*). The buffer memory is built from a two port Random Access Memory (RAM). Write controller is responsible for receiving the incoming packet and storing it in the free_space queue. The read controller is responsible for reading the granted packet from its associated queue. The Linked_List Update manages the linked list registers. Figure 3.12 shows the high level schematic of the input unit. The following is a discussion of the control registers used to control the buffer using linked list control.

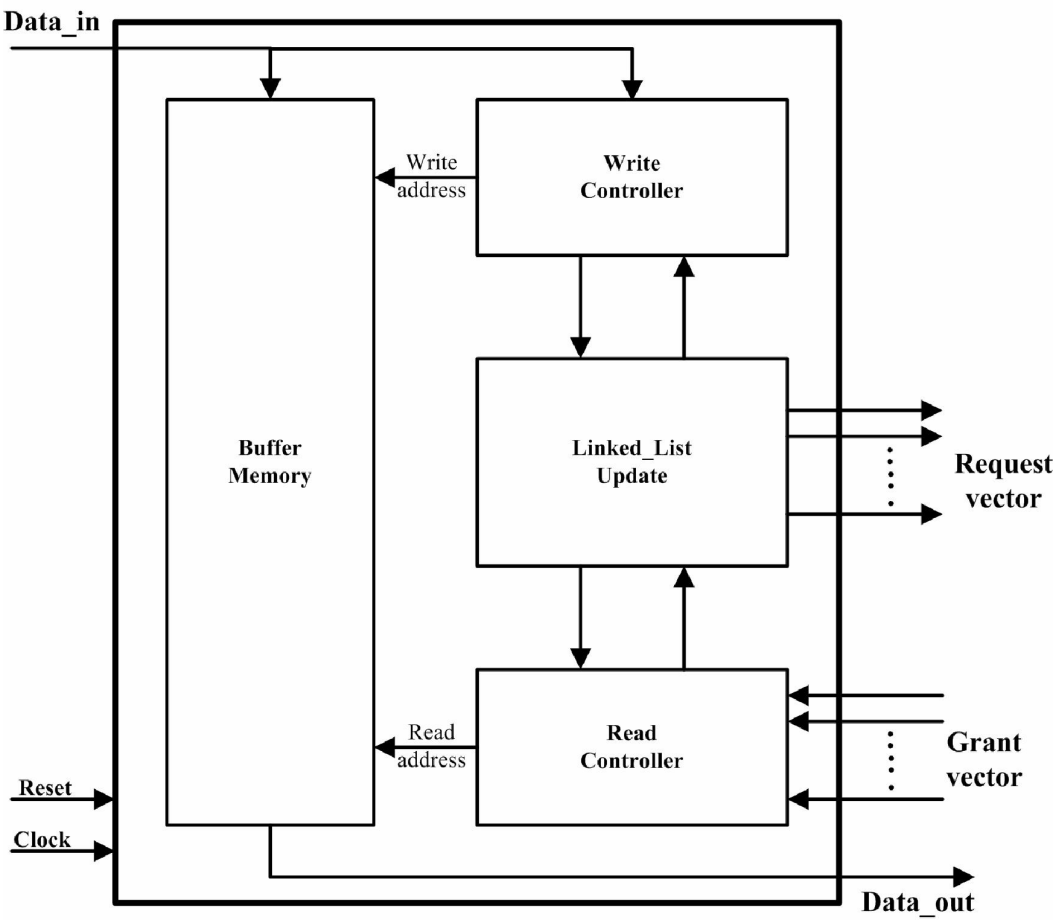


Figure **3**.12: VOQ input unit high level schematic.

Queue Empty flags:

It is a bit vector flags named "*queue_empty*" each associated to different virtual queue. If set to one, that indicates there are no packets in the virtual queue and reset to zero during writing of the first byte of the first packet. The default value is ones vector.

Request:

The request is an output bit vector signal. Each bit is set to one when the first byte of the first packet is written to the associated queue and reset to zero when last packet is read from the queue. It is simply the inverse of the queue_empty flags.

Next_registers:

Array of registers equal the number of blocks in the whole buffer. Each indicates the next place to the current block in the same queue.

Queue Head:

Array of registers hold the address of the first packet stored in its associated queue.

Queue Tail:

Array of registers that hold the address of the last packet stored in its associated queue.

Destination_port:

A register that holds the destination address of the incoming packet.

Queue_no:

An integer register that holds the destination address of incoming packet used to update the *queue_empty* flags

Counter_temp:

A counter variable used for counting the packet phits in both write and read controllers.

Write_address:

A register that holds the address of the current writing byte in memory.

Read_address:

A register that holds the address of the current reading byte from memory.

Address_signal:

A table that contains the destination addresses of all the packets in the buffer.

free_space_head:

A register that holds the address of the first packet in the free space queue.

free_space_tail:

A register that holds the address of the last packet in the free space queue.

free_space_empty:

A bit flag that sets to one when there are no packets in the free space queue.

Freespace_headtemp:

A temporary register that holds the address of the first packet in the free space queue while manipulating in the linked list update state machine.

Freespace_tailtemp:

A temporary register that holds the address of the last packet in the free space queue while manipulating in the linked list update state machine.

The following sections are describing in details the main components of the input unit.

3.3.2.4 The Buffering Memory

The input buffer consists of synchronous dual port memory. The dual port memory consists of an array of signal registers (D-flip flops) with a port for reading and another port for writing. There are two address lines in the design; the first is the write address (*Wr_address*) which determines where the data should be written, the other address is the read address (*Rd_Address*) which determines where the outgoing data are read from. There are separate enabling signals for writing (*wr_en*) and for reading (*rd_en*). The write and read operations is synchronized with the rising edge of the main clock. The dual port memory can manipulate writing incoming packet and reading outgoing packet in the same time on a separate data ports (*data_in*) for writing and (*data_out*) for reading.

The buffer in the case of VOQ is similar to the FIFO case and the schematic was presented in Figure 3.5. The buffer is divided into banks; each bank consists of a number of data_widths equal to the size of the packet where the data_width is general in the code and can be easily changed. The two-port means the RAM has separate read and write ports with associated read and write addresses. It was written in VHDL as an array of signals. It can be replaced by a ready block by the VLSI vendor. Figure 3.5 shows the schematic of the buffer memory.

3.3.2.5 Write Controller

The write controller is a state machine responsible for receiving the incoming packet and storing it in the free space queue. This state machine has three states operating at the rising edge of the main clock of the switch. All state transitions happen by a separate process called *Wr_state_change*.

The condition of the reset is checked at the start of the state machine. If the reset is active, the state machine does the following: disable the buffer write_en, resets the *counter_temp*, and resetting the state to state zero.

The state machine stays at state zero until the input start of frame (*sf_in*) of the incoming packet, (*sf_in* is sampled at the rising edge of the main clock of the switch). Upon the reception of start of frame (*sf_in*) signal, the state machine checks the state of the free space buffer. If the free space queue is empty; which means that no space for new packet; it drops the packet by keeping itself in state zero. If the free space queue is not empty, the packet is written to the head of the free space queue.

The storing (writing) address is calculated by appending (combining) both the *free_space_head* register as highest significant bits and the *counter_temp* as least significant bits. It saves the destination address to a separate place (*destination_port*) to be used by the linked list update state machine. It also changes its next state to state one.

The linked list update state machine updates the linked list registers at the time of state one. At state one, *counter_temp* is incremented, and the second phit is stored. The state machine goes to state two.

At state two, *counter_temp* is incremented, and the rest of the packet is stored then changes to state zero to continue receiving new packet. The state machine goes to state zero. Figure 3.13 shows the algorithmic state machine of the Write Controller.

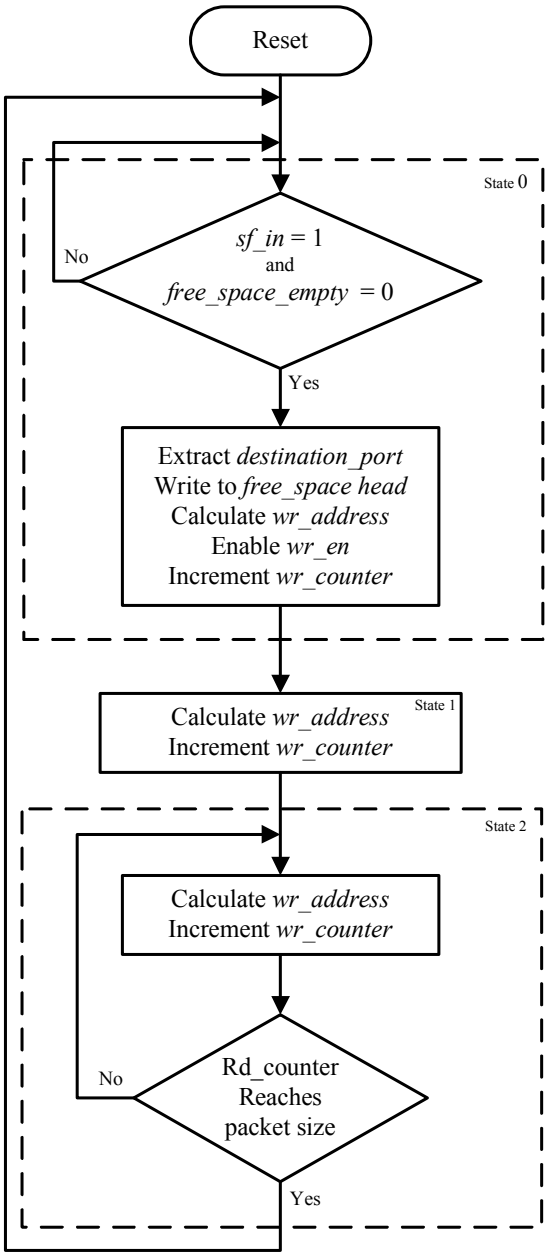


Figure 3.13: Algorithmic state machine of the VOQ write controller.

3.3.2.6 Read Controller

The read controller consists of a state machine with three states that operate at the rising edge of the main clock of the switch. All state transitions happen by a separate process called *Rd_state_change*. The read controller is a state machine responsible of transmitting the selected packet to the switching core after

reading it from its associated queue upon the grant signal. The state machine is synchronized with the rising edge of the system clock.

The condition of the reset is checked at the start of the state machine. If reset is active, three following actions are done; disables the *Rd_en*, resets *counter_temp*, and resetting the state to state zero.

The state machine stays at state zero until a grant signal arrives from the arbitration unit to allow the granted packet; which is the packet at the head of the queue; to cross to the switching core (the grant signal is sampled at the rising edge of the main clock of the switch).

Unless there is no active grant, the state machine stays in state zero. Upon detecting an active grant signal, the read controller calculates the granted *read_queue* from the grant vector signal. Then calculate the reading address by appending (combining) both the specified *queue_head* as highest significant bits and the *counter_temp* as least significant bits.

After the reading address is calculated, the buffer read enabling signal (*Rd_en*) is asserted and the first byte of the dequeued packet is read from the specified queue and put on the bus to the switching core at the rising edge of the clock by the buffer state machine.

At state one, the state machine raises the *sf_out* to logic 1, calculates the reading address, and reads the data from the buffer to output. Then it changes its state to state two. This state is used in the linked list update state machine to update the linked list registers.

At state two, the state machine resets the *sf_out* to logic zero, continues calculating the reading address and reading the data from the buffer to output until the end of the packet. Then it returns to state zero. Figure 3.14 shows the algorithmic state machine of the Read Controller.

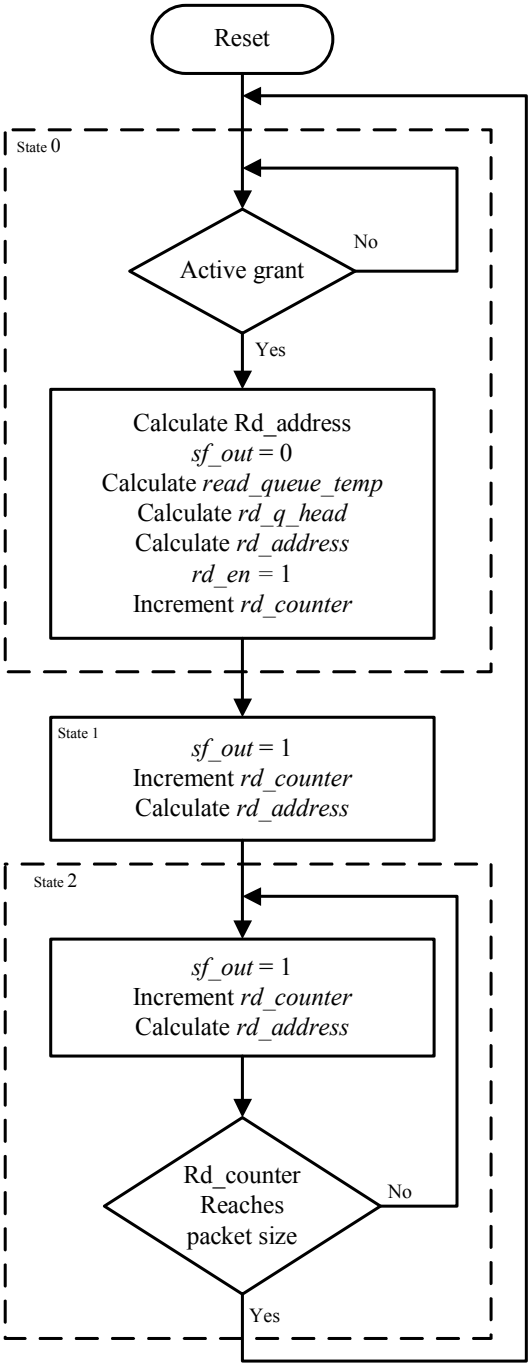


Figure **3**.14: Algorithmic state machine of the VOQ write controller.

3.3.2.7 Linked List Update

Linked list update is a state machine that controls moving blocks between free space queue and the other queues. It also manages the linked list registers. The state machine is synchronized with the rising edge of the system clock. It works actually upon state one of each of the writing and reading controllers. At resetting the state machine (rising the reset signal), it resets every queue head and tail to zero and every empty bit to one except the empty bit of the free space queue which is reset to zero. It also sets the next register of every block to the address of the following block.

Upon state one in the writing controller, it moves the block just received from the head of the free space queue to the tail of the specified queue. It firstly assigns the tail of a certain queue the address of the free space queue, it then checks if the certain queue was empty, in this case its head will be equal its tail. Otherwise, it updates the next register of the last tail of the certain queue by the address of the head of free space queue. It then removes that block from the head of free space queue. If the block was the last block in the free space queue, it will raise the empty flag of free space queue. Otherwise, it will assign a new head to the free space queue by the next register of the old head. Upon state one in the read controller, it removes the block from the head of the certain queue and appends it to the tail of the free space queue. It firstly checks whether that block is the last block in the certain queue, it then raises the empty flag of that queue. Otherwise it assigns a new head to the certain queue by the next register of the old head. It then checks whether the free space queue is empty. If so, it resets the empty flag and assigns its head and tail by the address of the certain queue old head. Otherwise, it assigns the free space tail by the address of the certain queue old head and updates the next register of old tail by the new tail. Figure 3.15 shows the flow chart of the Linked list update state machine. Appendix A gives a practical example of linked list control.

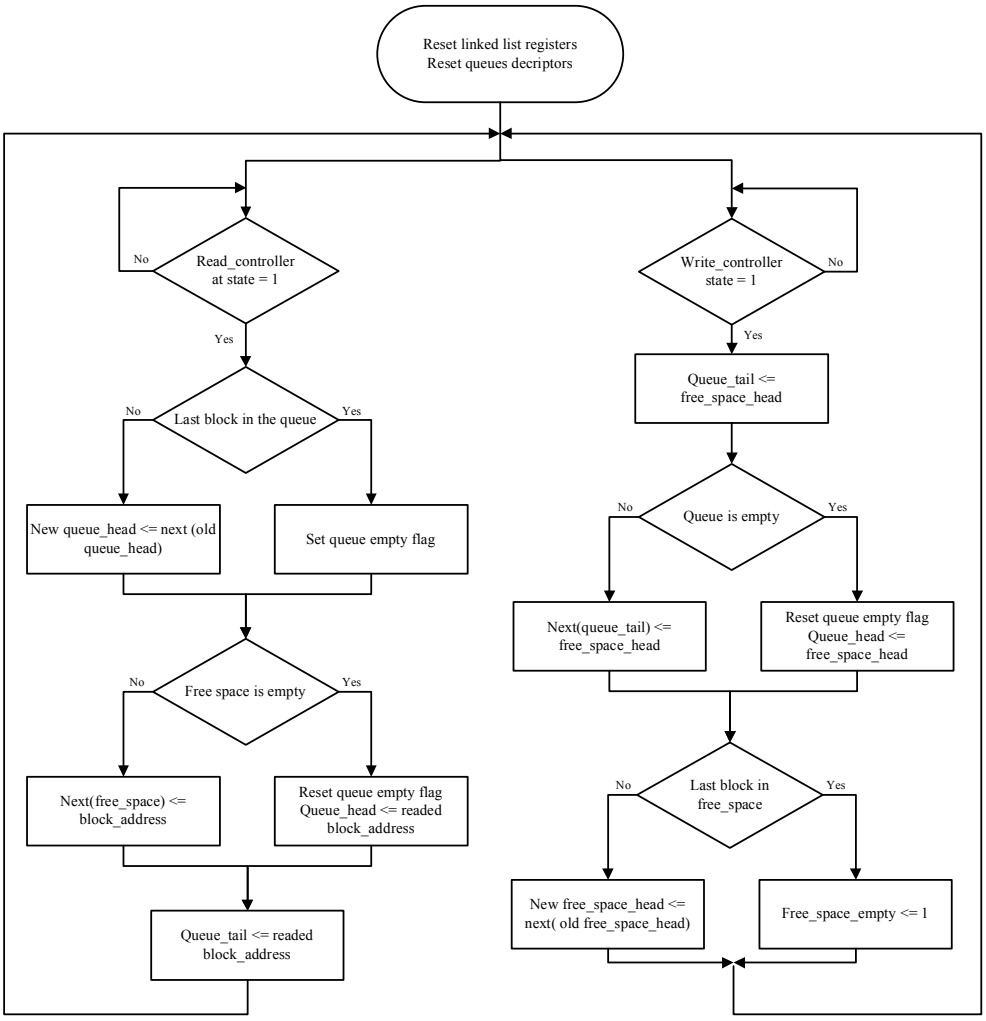


Figure **3**.15: Flow chart of the linked list update state machine.

3.4 Switching Cores

The switching core is a network of connection ON/OFF switches that can be closed (ON) to route the data from the input to its destination at the output. The widely used switch cores are Crossbar, Fully-Connected and Batcher-Banyan switches. The following subsections discuss the implemented switching cores in this work.

3.4.1 Crossbar Switching Core

In this work, large 128×128 crossbar switching core is implemented. The implemented Crossbar switching core consists of 128 horizontal buses holding the 128 inputs. Each bus is *bus_width*, which is 8 bits in the design. On each bus, 128 crosspoints are connected to vertical buses and each crosspoint is dedicated to one output. The crosspoint is 8 bits ON/OFF switches, each ON/OFF bit switch is constructed by a 2-input AND gate; the first input of the AND gate is the data input bit, whereas the second input is the control signal. An OR gate at each output is used to collect all outputs of the crosspoints dedicated to the same output bit. There is one control bit for all the bit width of the bus. Figure 3.16 shows both the crosspoint theoretical symbol and its hardware implementation by gates. Figure 3.17 shows the implementation of 4×4 Crossbar switching core.

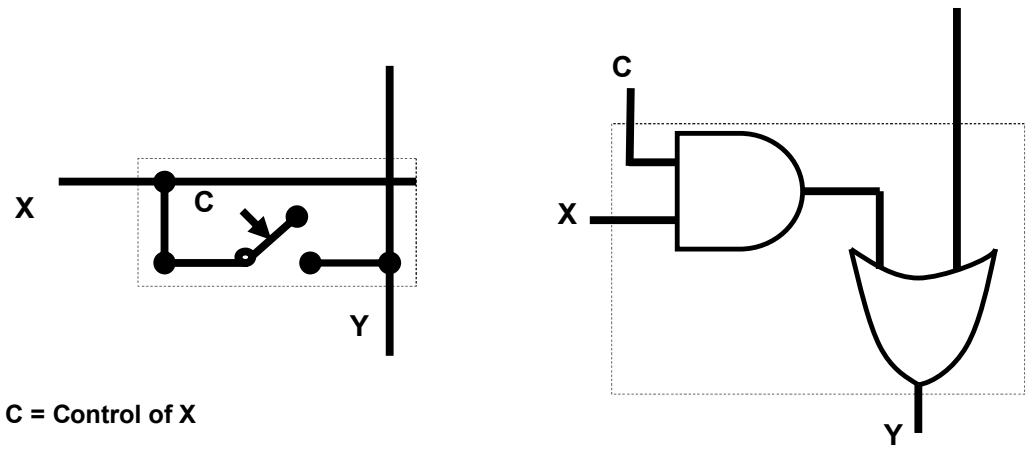


Figure **3**.16: Both the crosspoint theoretical symbol the its hardware gate implementation.

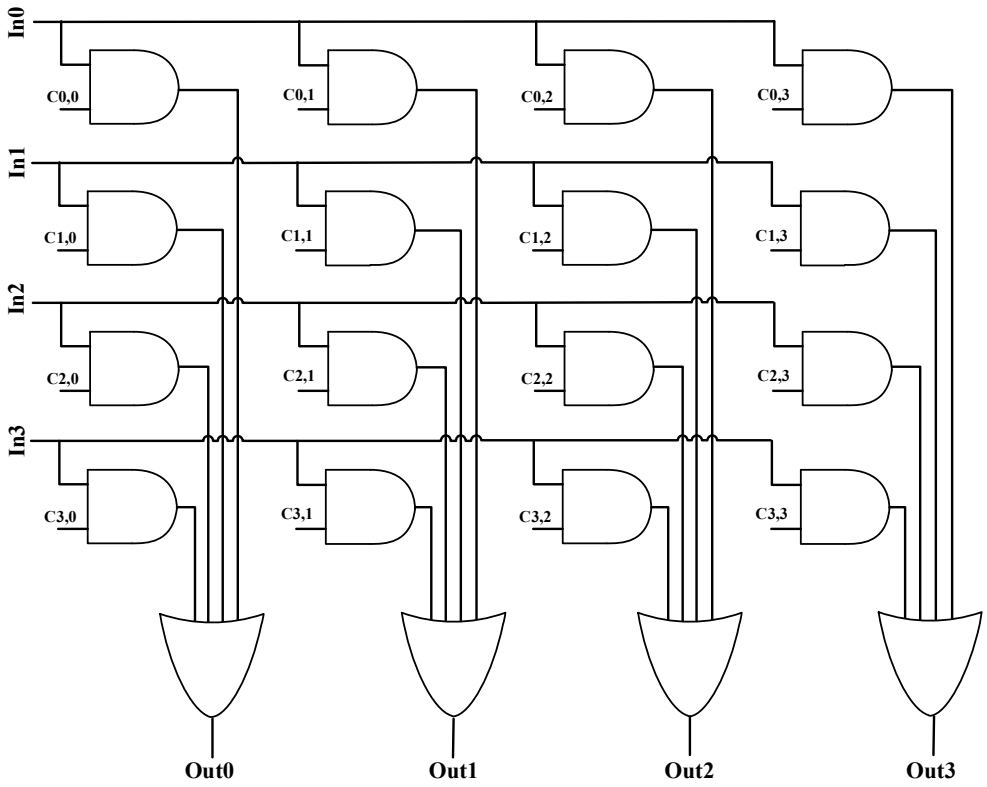


Figure **3**.17: A 4×4 crossbar switching core.

3.4.2 Banyan and Batcher-Banyan Switching Cores

In this work, 128×128 Batcher-Banyan switching core are implemented. The following subsections discuss the hardware implementation of both building blocks of Batcher-Banyan switching core.

3.4.2.1 Hardware Implementation of Banyan Switching Network

The 128×128 Banyan switching network consists of seven stages. Each stage contains 64 nodes, each node is a 2×2 self routing switch.

The Banyan network is connected as narrow-sense banyan network [42]. Each node includes two multiplexers and a controller for the self routing property, the controller controls the two multiplexers according to a certain bit of the

destination address in the header of the input packets according the stage where the node belongs.

Figure 3.18 shows the detailed schematic diagram of the switching node. The main components of the switching node are two multiplexers used to multiplex the input data (in our case the input data are one phit (eight bits) and *sf_in*). The multiplexers are controlled by a self routing controller depending on the address of the input packets.

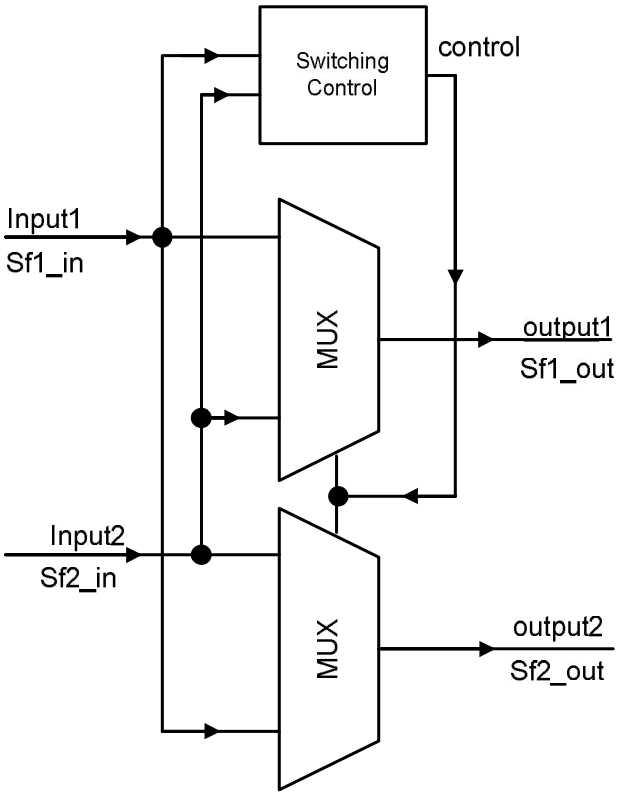


Figure **3**.18: Schematic diagram of the Banyan node.

The control circuit samples the incoming start of frame (*sf1-in*, *sf2_in*). If any of them is active, the self routing controller uses a specific routing bit in the destination address which depends on which stage the node is located. For example, the nodes in the first stage check the first bit in the destination address. Nodes in the second stage check the second bit and so on. If both inputs of the switching node are containing start of frame (active), checking the routing bit in

the first input is enough because the routing bit in the second input must be its inverse. If one of the inputs is only active, its routing bit is checked. The 128×128 narrow sense Banyan switching node connections are illustrated in Figure 3.19.

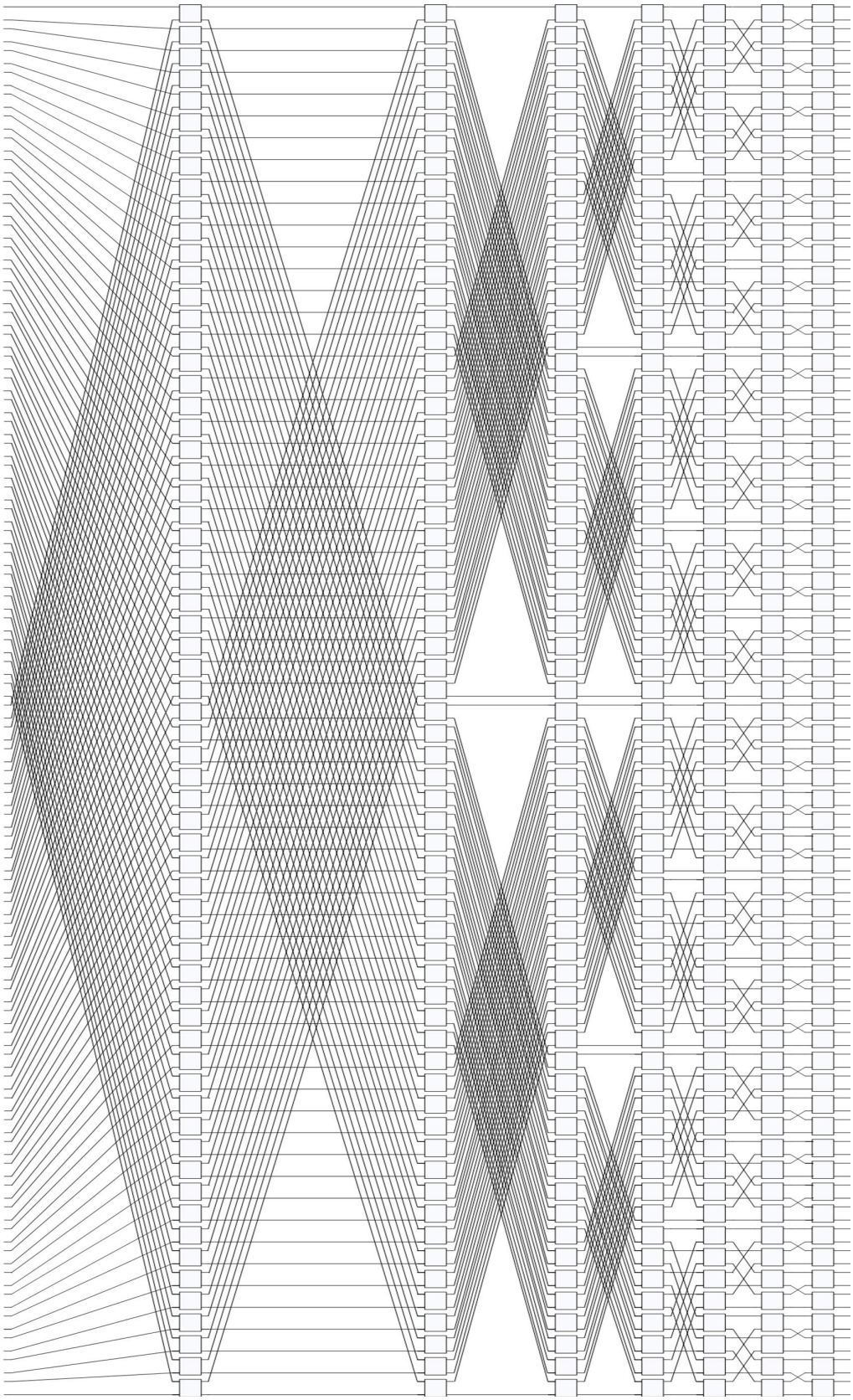


Figure **3**.19: A 128×128 narrow sense Banyan connection network.

3.4.2.2 Hardware Implementation of Batcher Sorting Network

The Batcher sorting network consists of seven stages; each stage contains a different combination of two different types of self-routing 2×2 sorting nodes in progression manner. Each type of sorting nodes sorts the two input packets in ascending or descending manner according to its type. Each sorting node includes two multiplexers and a controller for the sorting and self routing property, the controller controls the two multiplexers according to the destination address in the header of the input packet.

Figure 3.20 shows the detailed schematic diagram of the sorting elements. The main component of the switching node is two multiplexers used to multiplex the input data (in our case the input data are one phit (eight bits) and *sf_in*). The multiplexers are controlled by a sorting control unit which depends on the destination addresses of the input packets.

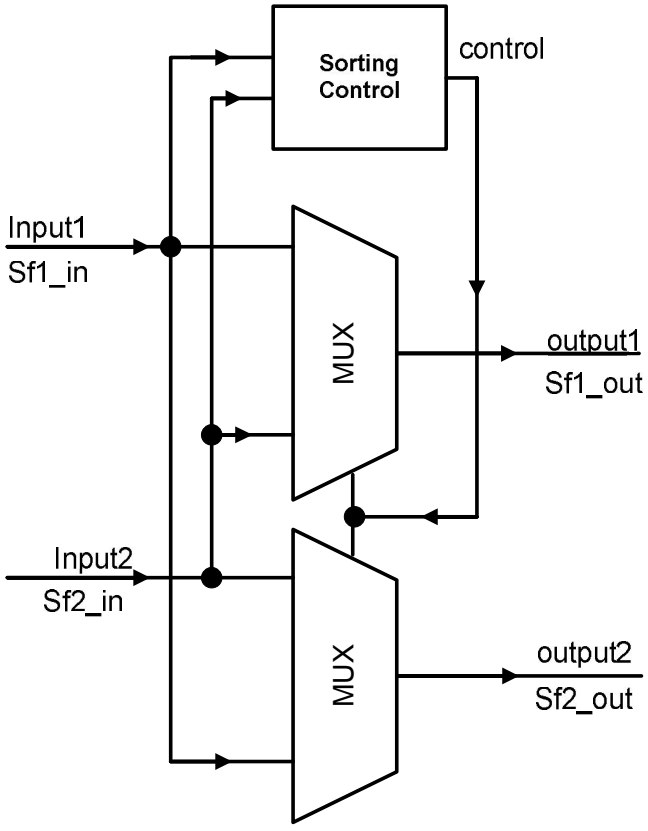


Figure **3**.20: Schematic of the Batcher sorting element.

The sorting control circuit samples the incoming start of frame (*sf1-in*, *sf2_in*). If any of them is active, the sorting controller uses the full destination address to sort ascending or descending. If both inputs of the sorting node are containing start of frame (active), sorting is held between the destination addresses. If only one input is active, it is routed to the upper in case of Batcher_descend node or lower port in case of Batcher_ascend node. The 128×128 Batcher sorting network is shown in Figure 3.21.

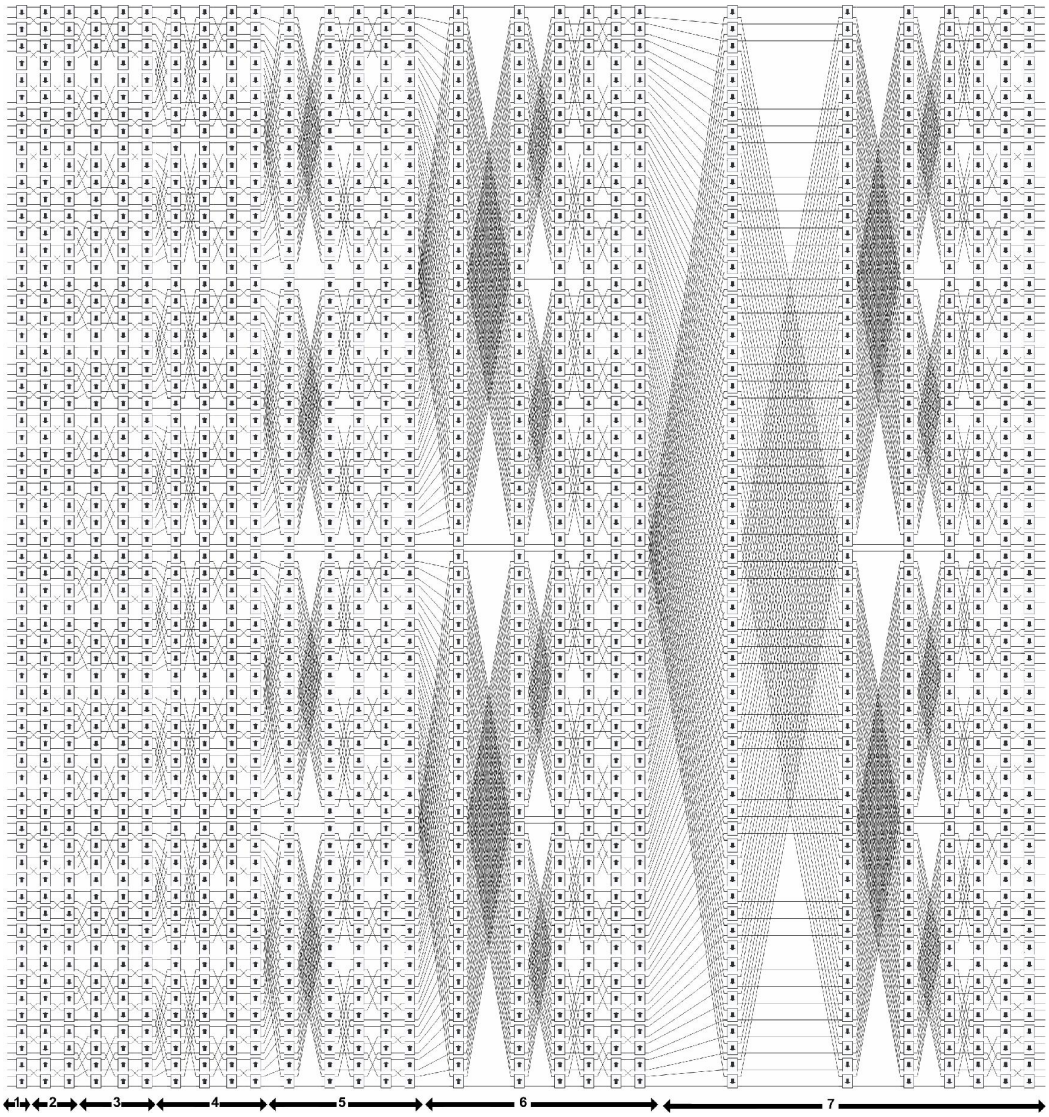


Figure **3**.21: A 128×128 Batcher sorting network.

3.5 Arbitration Unit

For the FIFO buffering strategy, we implemented an arbitration algorithm called Ring-Reservation which is the best candidate for its simplicity, low area cost and low power consumption. If VOQ buffering strategy is used, a fast and intelligent arbitration mechanism is required. In VOQ, the role of arbitration becomes more complex consumes more power because all of the logical queues in each input unit buffer (virtual queues) will issue a separate request and wait for a separate grant. So, the total number of requests will be N^2 (there are N input units and each input unit contains N virtual queues), all need to be arbitrated in each time slot.

The arbiter must also provide N^2 grants for the virtual queues. There are many theoretical arbitration algorithms for virtual output queuing like Maximum Size Matching, Maximum Weight Matching, Oldest Cell First (OCF), Longest Port First (LPF), Parallel Iterative Matching (PIM), Round Robin Matching (RRM), iSLIP, etc. Most of them are hard to implement in hardware or even impossible. Thus, the propagation arbiters are the only applicable arbiter for the VOQ input buffering [44][45]. The Diagonal Propagation Arbiter (DPA) belongs to the propagation arbiter's family.

In this work both Ring-Reservation and DPA are implemented to resolve the destination contention in the presented designs in this thesis.

3.5.1 Ring-Reservation

Bingham *et al.* presented an arbitration unit for the FIFO input buffered Batcher-Banyan switches called Ring-Reservation arbitration unit [46]. The Ring-Reservation keeps track of repeated requests (requests with repeated destination address) during scanning each request from the input units, and then issues the grants to a group of none repeated requests. The Ring-Reservation consists of one main control unit called Ring Head End (RHE) and many small

controllers called Cell Switch Interfaces (CSI) where each CSI is dedicated to one input unit. Every CSI compares the request address from its dedicated input unit with all available addresses of each output port in circulation manner and reserves it if the matched destination address has not been reserved for another input before in the arbitration cycle.

The arbitration cycle starts by scanning for requests along with issuing the grants and finishes by transmitting the granted packets. For fairness, a round robin rotation controlled by the RHE changes the start of circulation each arbitration cycle to change the highest priority CSI. After the reservation cycle completes, every CSI issues grant signal for the reserved input port in the grant phase. RHE controls the start and the finish of the process as well as the timing of circulation and issuing grants. Figure 3.22 shows the Ring Reservation unit with respect to the input units and the Banyan switch. The following subsections discuss the implementation details of both RHE and CSI.

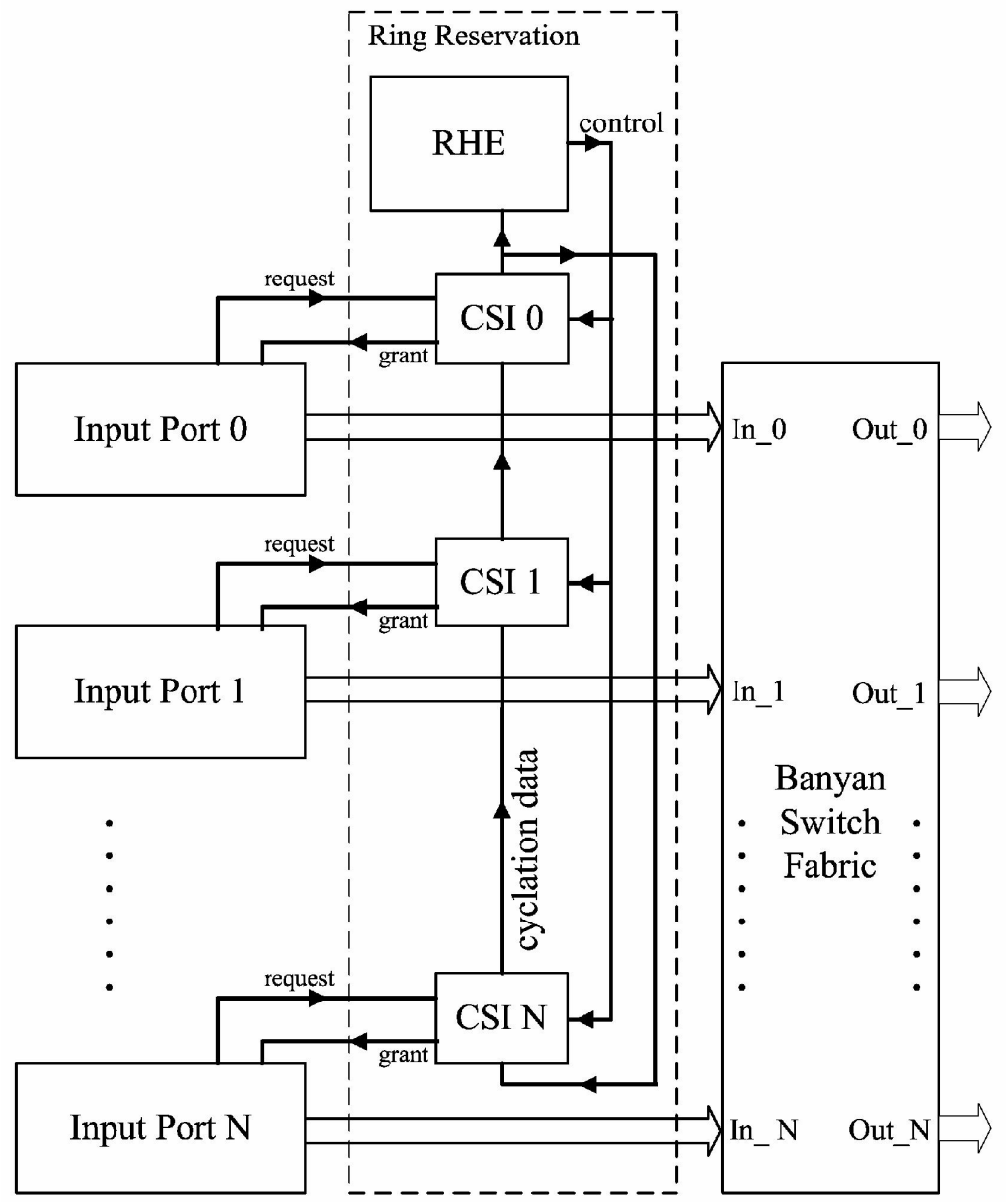


Figure **3**.22: The ring reservation unit with respect to the input units and the Banyan Switch.

3.5.1.1 CSI

The Cell Switch Interface (CSI) is a state machine synchronized with the system by clock and reset signals. It communicates with both its dedicated input unit and the RHE with signals discussed as follow:

Request: is the request input signal from the associated input unit.

Address_in: the address input associated to the request from the input unit.

Grant_out: grant output signal to the input unit.

Grant_flag: internal grant issued at the end of the arbitration cycle.

R_data_out: data output path that rotates the addresses to the following CSI.

Token_out: token output signal to the following CSI shows whether the address on the *R_data_out* is previously reserved or not.

R_data_in: data input path that transfers the rotated addresses from the previous CSI.

Token_in: token input signal from the previous CSI shows whether the address on the *R_data_in* is previously reserved or not.

Rotate: command input signal from the RHE sampled at the rising edge of the clock to force the CSI to progress in the arbitration cycle by rotating the addresses and scanning the request.

Position: location of the CSI or the number of the input unit associated.

Reset and *Clock*: router reset and clock inputs.

Figure 3.23 shows the block diagram of the CSI.

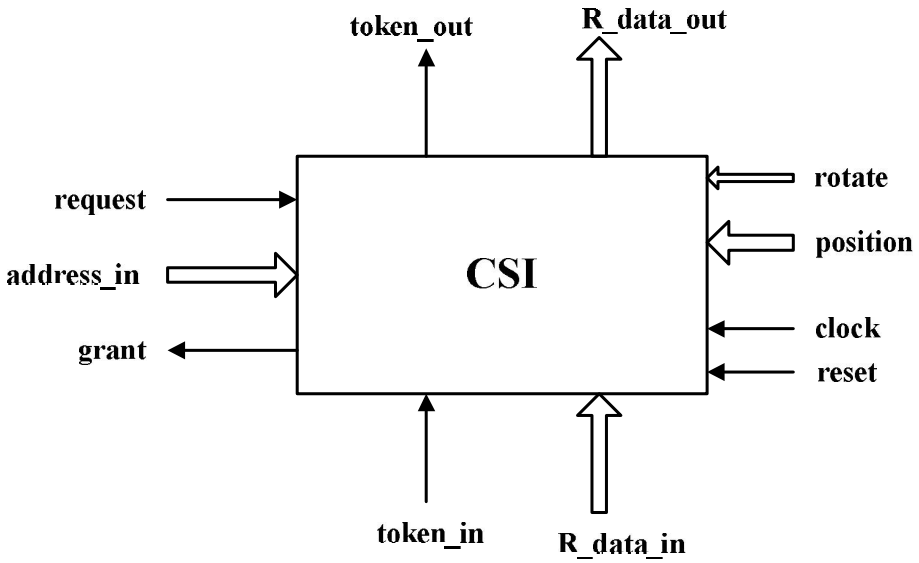


Figure **3**.23: Block diagram of the CSI.

CSI is a state machine that operates at the rising edge of the clock. The reset is checked at the beginning of the state machine. At reset condition, the CSI state machine resets the *grant* and *token_out* and saves the position input.

There are one state machines operate inside the CSI "*csi_rd*". It samples each of the *R_data_in, token_in*, and *rotate* at the rising edge of the main clock of the switch to a temporary registers *R_data_signal, token_signal,* and *rotate_signal*.

At normal operation, the state machine checks the input command signal "*rotate*" from the RHE, the command rotate; which is a two bit signal; has four commands. The first rotate command is "00" which is the default state when there is no request to the switch and after the reservation circulation. It forces the CSI to reset the *grant* signal.

The second command "11" forces the CSI to compare the temporary register *R_data_signal* with the requested *address_in*. If a match occurs while the associated token *token_signal* register is zero (the address is not previously reserved), it sets an internal *grant_flag* meaning that the associated input port has been granted then sets the *token_out*. If no match occurred, the *token_out* is reset and the internal grant flag remains as before. The CSI then copies the *R_data_sigbal* to the *R_data_out* port. This command is repeated along the reservation cycle.

The third command "10" tells the CSI to issue the grant if the internal *grant_flag* is set, and then resets the *token_out* and copies the *R_data_signal* to the *R_data_out* port to change the initial *R_data_signal* in the next reservation cycle for fairness among all input ports. Figure 3.24 shows the flow chart of the CSI state machine.

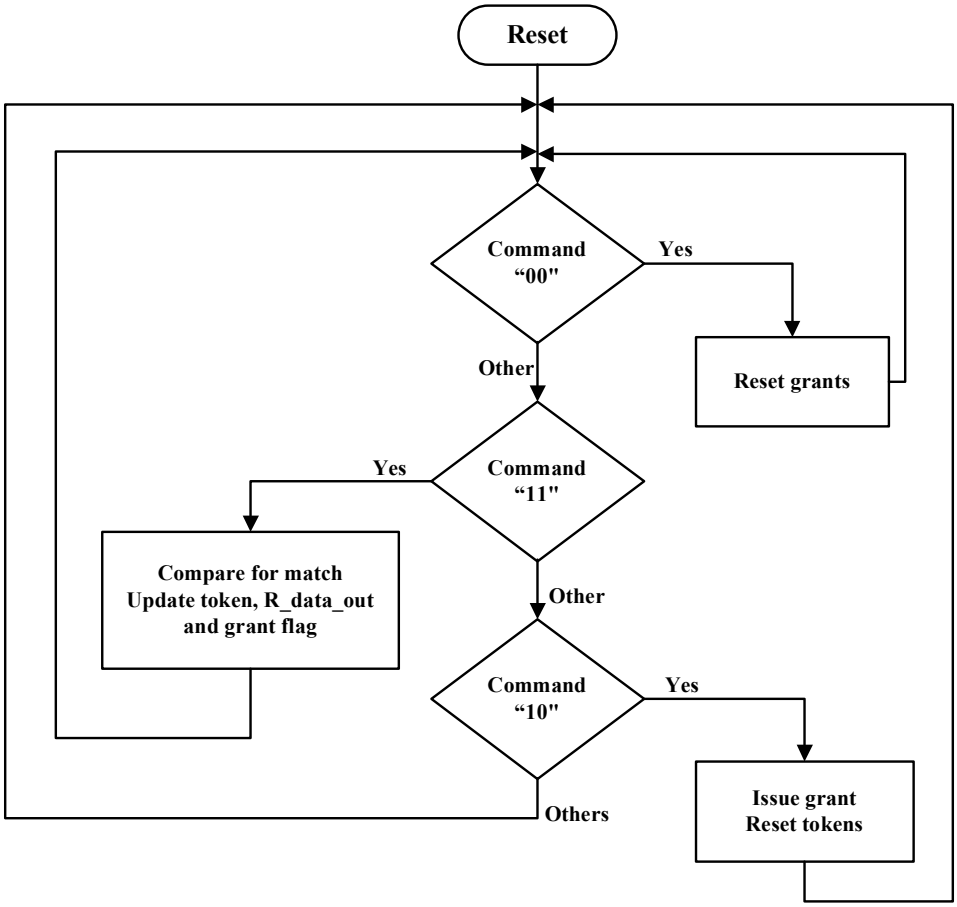


Figure **3**.24: Flow chart of the CSI state machine.

3.5.1.2 RHE

The Ring Head End (RHE) is a state machine synchronized with the system by clock and reset signals. Figure 3.25 shows the block diagram of the RHE. RHE communicates all input units and all the CSI’s with signals discussed as follow:

Global_request: if there is at least one request from any input unit is active, *global_request* comes from the OR operation of all requests.

Rotate: command output signal to all the CSI’s.

Reset and *Clock*: router reset and clock inputs.

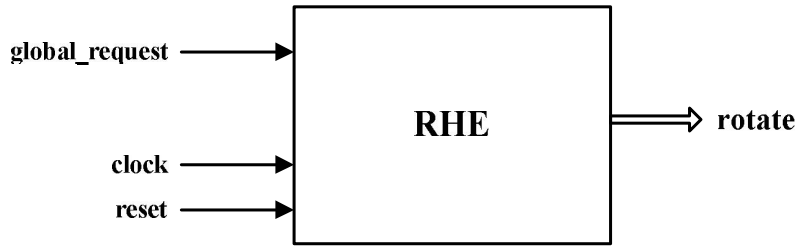


Figure **3**.25: Block diagram of RHE.

The RHE state machine has three states and operates at the rising edge of the clock. At reset state, it forces the rotate command output to "00" and forces the next state to state zero.

At state zero, the *global_request_signal* is checked. If active (set to one), it starts an internal counter then it changes the command output to "11" on the *rotate* output.

At state one, the reservation cycle is in process, so state one remains until the counter overflows to zero, it takes *N* cycles of the main switch clock, where *N* is the number of input/output (size) of the switch. After the reservation cycle completes, a "10" command is delivered at the rotate port then it forces the next state to state two.

At state two, command "00" is delivered to the rotate port and forces the next state to zero. The state transitions of the main state machine are manipulated by the separate fourth state machine called state_change. Figure 3.26 shows the flow chart of the RHE.

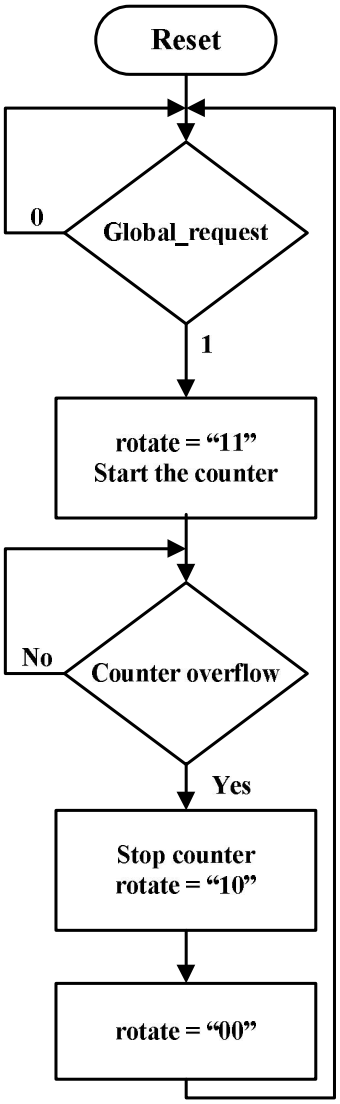


Figure **3**.26: RHE flow chart.

3.5.1.3 Ring Reservation Unit

The detailed connection of the Ring reservation unit for 4×4 switch is shown in Figure 3.27. This scheme of arbitration will become the bottleneck when the number of ports of the switch is large. However, by arbitrarily setting the appropriate values for the counters prior to the arbitration, this scheme provides fairness among the input ports. Another advantage of this scheme is that it can be employed at the input of any type of switch fabric. Appendix B gives a detailed example for the operation of the ring reservation arbiter.

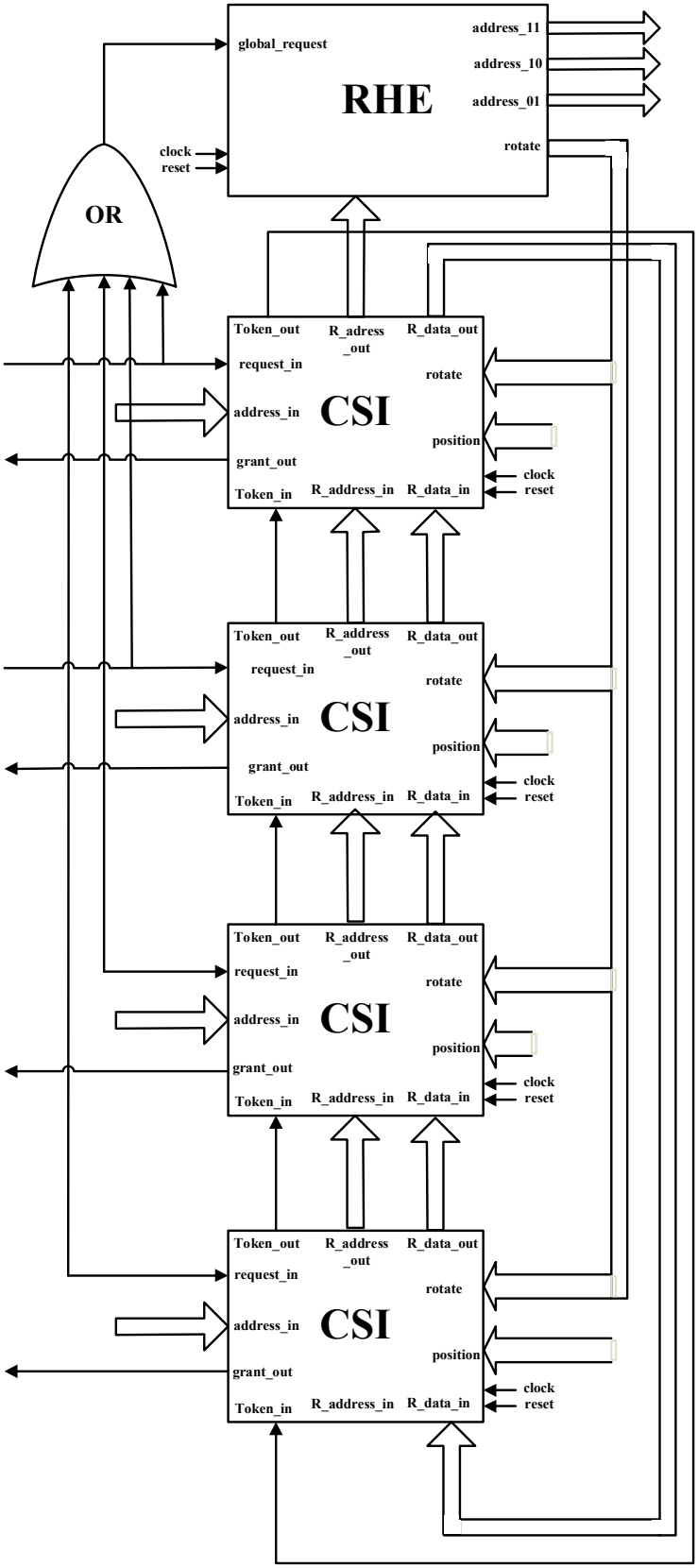


Figure **3**.27: Detailed connection diagram of ring reservation unit for a 4×4 switch.

3.5.2 The Diagonal Propagation Arbiter

Hurt et al. presented the Diagonal Propagation Arbiter (DPA) in [44]. DPA is an arbiter unit for VOQ input buffering strategy that provides a simple and fast arbitration algorithm that can be easily implemented in hardware. The most interesting advantage of this algorithm is that it can be designed using combinational logic (conventional standard cells without the need of flip-flops). DPA is based on a two-dimensional ripple carry arbiter architecture proposed in [47], in addition to a round-robin scheme to rotate the priorities.

3.5.2.1 Two-dimensional ripple carry arbiter

For each router, there are N ports and for each port, there are N virtual output queues. Each queue issues a separate request and waits a separate grant. So, the total number of requests is N^2 and the total number of grants is N^2. The two-dimensional ripple carry arbiter consists of N^2 arbitration cells connected with each other in a mesh network. Each cell is responsible for receiving a request and issuing a grant from its corresponding virtual output queue. Each arbitration cell takes one request and output one grant. The arbitration cell also has four control signals; the four signals are divided into two inputs from its upper and left neighbors and two outputs signals for its lower and right neighbors. The block diagram of the arbitration cell is shown in Figure 3.28. The practical schematic of the arbitration cell is shown in Figure 3.29. The cell placement for the original 4×4 two-dimensional ripple carry arbiter which is the original architecture of our arbiter is shown in Figure 3.30.

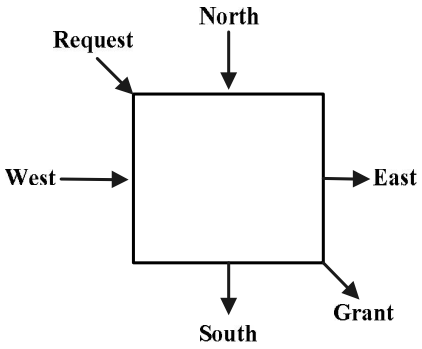


Figure **3**.28: Block diagram of the arbitration cell.

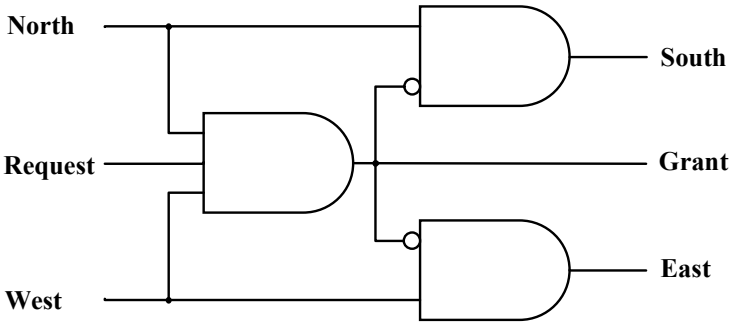


Figure **3**.29: Practical implementation of the arbitration cell logic.

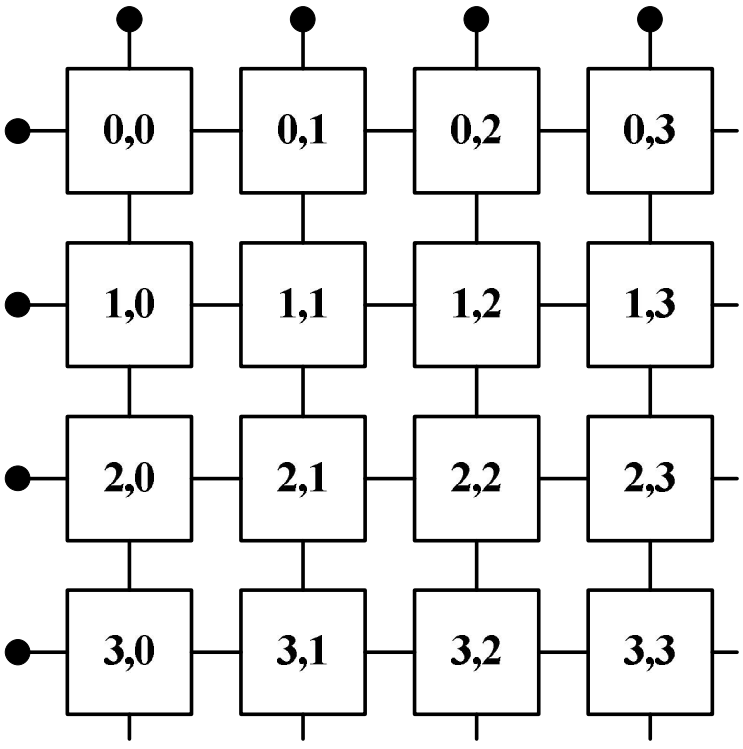


Figure **3**.30: Placement of the arbitration cells in a 4×4 two-dimensional ripple carry arbiter.

In the two-dimensional ripple carry arbiter architecture; there are n×n arbitration cells arranged in mesh network. From Figure 3.30, the two numbers separated by a comma inside each arbitration cell (*i,j*)) corresponding to row *i* and column *j* in the arbiter. The two numbers are also representing the input port (to the left) and the virtual queue (to the right). For example (2,3) represents the port number 2 and the virtual queue number 3. We notice from the figure that each west input signal is connected to the east output signal from the neighbor to the right except the first which is connected to the logic high and the east output signal of the last column is free. Each north input signal is connected to the south output signal from the above neighbor except the upper row where its north input is connected to logic high and the south output of the lower row is free.

Each cell can issue a grant if and only if there is no other grant in the same column and row that means no multiple grants are issued to the same input unit and no more than one grant is issued to the same output port. That prevents the occurrence of resource conflicts (contentions).

Arbitration algorithm

Every arbitration cell is responsible for issuing the grant to the request from its specified virtual queue. Each cell ensures that the higher priority cells in its row and column did not issue a grant before it issues its virtual queue. It takes the decision of issuing the grant after checking that no grant has been issued from higher priority cells in its row or column.

The arbitration cycle starts at the top left cell which is the higher priority cell in the arbiter. The arbitration process propagates diagonally towards the bottom right cell which is the least priority. Every cell in a diagonal has less priority than the cells in the previous diagonal.

Each cell receives a request $R(i,j)$ and issues a grant $G(i,j)$ to its corresponding virtual queue j in the input port i. At every arbitration cycle, at most only one virtual queue in each input port must be granted and at most only one packet must be granted to one output port. So, there must be at most only one grant issued from a single row in each arbitration cycle, and at most only one grant issued from a single column according to the following equation.

$$\sum_{i=1}^{i=N} G(i,j) \le 1 \quad \text{for } 1 < \text{j} < \text{N}\,, \quad \sum_{j=1}^{j=N} G(i,j) \le 1 \quad \text{for } 1 < \text{i} < \text{N}$$

So, if cell (i,j) issued a grant, it prevents any lower priority cell in column j (cells below row i) and any lower priority cell in row i (cells right of column j) from issuing a grant.

If an arbitration cell (in row i, and column j) received a request $R(i,j)$, it checks the north and west signals coming from the south (same column j) and east (same row i) outputs from its top and left higher priority cells. If both of them are not asserted (pulled down), it issues a grant $G(i,j)$. Once the cell issued a grant, it prevents any cell in column j below row i and any cell in row i right of column j from being granted by asserting both its south and east output signals. If any or both of them are asserted (pulled down), it rejects the request and doesn't issue a grant then assert the east or south or both signals if the west or north or both were previously asserted respectively. Appendix C gives a detailed arbitration cycle example

Fairness

In the two-dimensional ripple carry arbiter architecture, the top left cell (cell (0,0) in our example) always takes the highest priority over the rest of the cells. Cells in the second diagonal takes less than cell (0,0) and higher than the following diagonals and so fourth. This is the drawback of the two-dimentional ripple carry arbiter which leads to unfair arbitration and this is too bad.

To ensure fairness, the highest priority cell (top left) must be changed in each arbitration cycle by adding any priority rotation. So, Hurt et al [44] proposed two amendments using round-robin scheme to rotate the priorities. The first architecture is called the Rectilinear Propagation Arbiter (RPA) while the second architecture is the Diagonal Propagation Arbiter (DPA). Both of them are discussed in the following sections.

3.5.2.2 Rectilinear Propagation Arbiter (RPA)

To achieve the fairness, RPA replicates the basic two-dimensional ripple carry arbiter design in both the horizontal and the vertical directions and uses two priority vectors to activate cells inside a window sliding in both horizontal and vertical directions. That changes the highest priority among all arbitration cells instead of only cell (0,0). The sliding window is of the size $N \times N$ where N is the size of the switch (number of input/output). The cells inside the window are activated and the cells outside the window are deactivated. For $N \times N$ ports, there are $2N$-1 rows and $2N$-1 columns. The sliding window changes its position every arbitration process. Figure 3.31 shows an example of 4×4 RPA consisting of 7×7 arbitration cells. From the figure, the sliding window gives the cell labeled (1,2) the highest priority. The sliding window can move in any direction using two priority vectors X and Y. The priority vectors X and Y are $2N$-1 bit vectors changing every arbitration process. In any arbitration process, only N bits in the vectors are active (logic one) while others are deactivated (logic zero). In our example of 4×4, and in the case in the figure, X= 0011110 and Y= 0111100.

One of the priority vectors rotates every arbitration cycle from 1111000 to 0001111 then resets at the following cycle to 1111000. The other priority vector rotates the same manner but every reset of the first priority vector. This can be designed using loadable circular shift registers to provide a round robin scheme for fair arbitration.

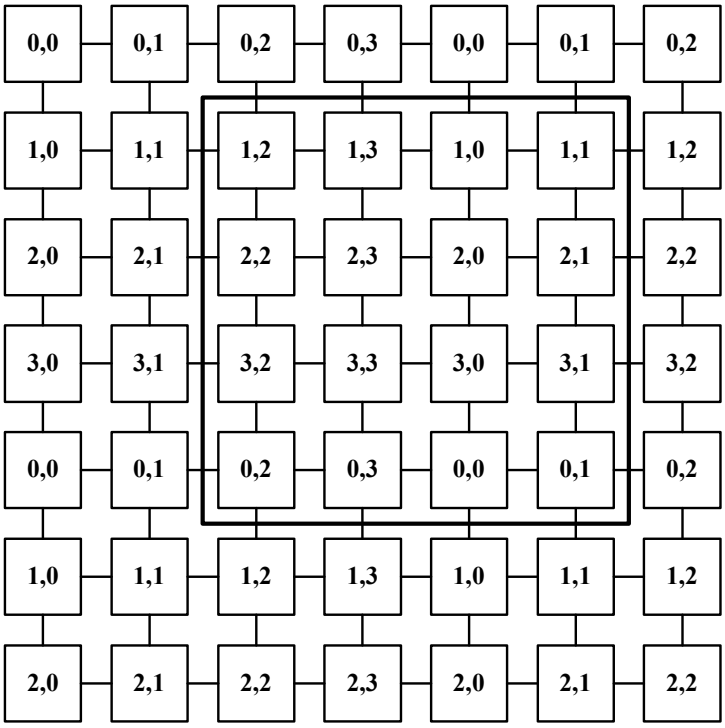


Figure **3**.31: A 4×4 RPA.

The arbitration cell must be modified to be able to be activated and deactivated by the priority vectors by gating the request signal entering to the cell by both the priority vectors. Figure 3.32 shows the RPA modified arbitration cell. The mask in Figure 3.32 comes from ANDing the two priority vectors *X* and *Y*.

The delay of the arbitration cycle is the time taken by signals to propagate from the top left arbitration cell to the bottom right cell. There are 2*N*-1 diagonals and each cell delay is D time so the total is

$$(2N\text{-}1)\mathrm{D} + 2\mathrm{t}$$

Where *N* is the number of ports, D is the delay of the original arbitration cell, and 2t is the delay of two AND gates activating the cell.

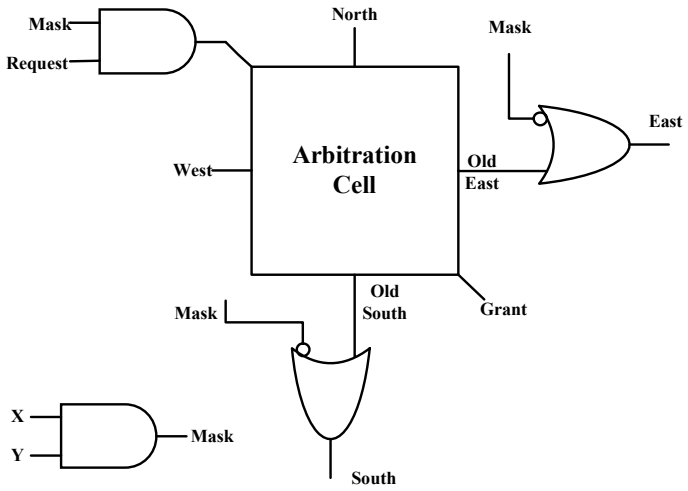


Figure **3**.32: The RPA modified arbitration cell.

3.5.2.3 Diagonal Propagation Arbiter (DPA)

The second architecture is the DPA. It consists of putting independent cells in one diagonal and replicating the basic cells only in the vertical directions and using only one priority vector to activate cells inside a window sliding in the vertical direction.

Figure 3.33 shows an example of 4×4 DPA. The sliding window moves vertically (up or down) instead of two directions as in RPA, that save the effort of using two priority vectors to only one vector. The bolded lined region is the activated sliding window cells by the priority vector. The priority vector according to the figure will be P= 0011110 which also rotates every arbitration cycle from 1111000 to 0001111 then resets to 1111000.

The arbitration cell must be modified by gating the request entering to the cell by the priority vector. Figure 3.34 shows the modified arbitration cell for the DPA, it shows gating the original cell by the generated mask which replaces AND gating both priority vectors used in RPA by a wire.

DPA shows better delay due to using lower number of cells than the RPA and hence lower area on silicon. The delay will be reduced due to the reduction of the number of diagonals the signal must propagate.

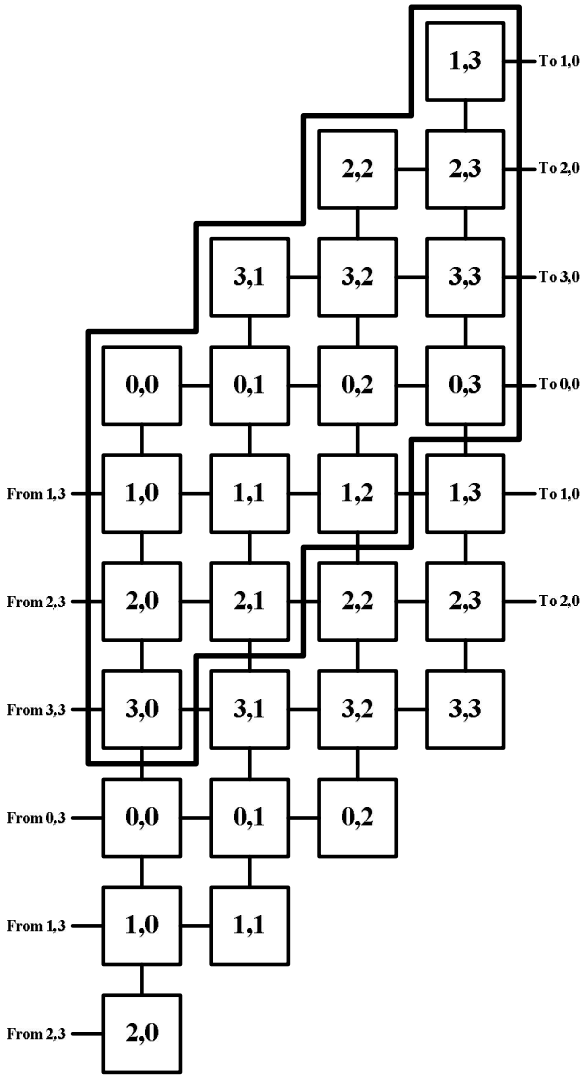


Figure 3.33: A 4×4 diagonal propagation arbiter DPA.

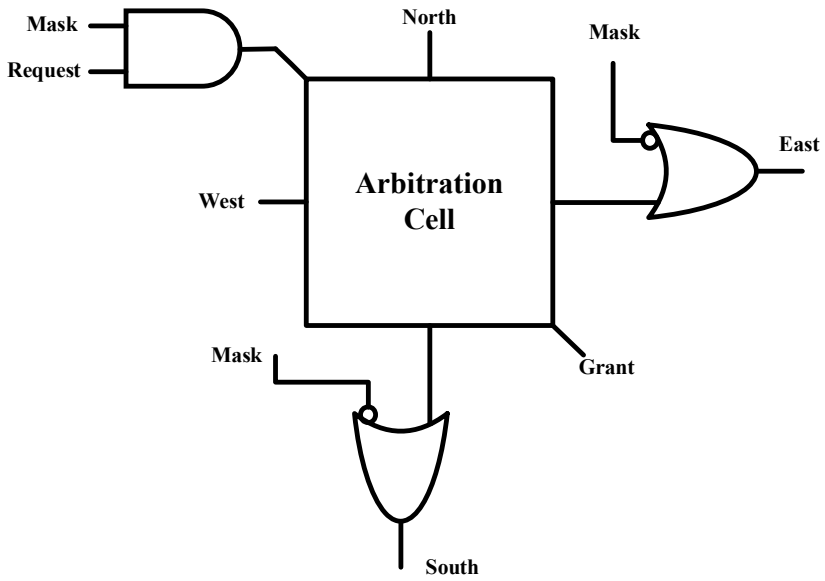


Figure **3**.34: DPA arbitration cell

The total delay will be

$$ND + t$$

Where N is the number of input/output ports, D is the delay of the original arbitration cell, and t is the delay of the AND gate on the request signal.

3.6 Chapter Summary

This chapter discussed the detailed design and implementation of all our routers building blocks. The first section discussed the input unit. Two buffering strategies are implemented, the FIFO which is the simple and direct way, and VOQ which is more complex. The second section discussed two types of the switching cores; Crossbar which is the most famous and Batcher-Banyan which is one of the multi-stage switching cores. The third section discussed two arbitration units; Ring Reservation and the Propagation Arbiters each arbiter is suitable for different buffering strategy discussed earlier.

Chapter 4

The Simulation Environment

In order to test the behavior of the proposed routers, they must be put in an environment resembling the real environment on a chip surrounded with many modules generating messages and waiting response. So, an environment that consists of two components is developed. The first component is the load generator and it is attached to the input ports of the router responsible of generating random packets resembling the packets coming from surrounding modules on the chip. The second component is the packet counter and it is attached to the output ports of the router responsible for evaluating the performance of the router.

4.1 The Load Generator

The load generator is responsible for generating packets with random destination addresses with uniform random distribution. It also assigns a time stamp to each packet in order to calculate the average delay at the data calculator. The load generator consists of a main controller and supporter components, all are discussed in the following subsections.

4.1.1 Switch Clock Generator

The switch clock generator is a state machine named *switch_clock* used to reduce the input clock to the router (*sw_clk*) where the main higher clock is used in the Pseudo random generator to complete generating 128 random destination addresses. The test environment operates on four times the router clock.

4.1.2 Pseudo Random Generator

The Pseudo random generator is responsible for generating 128 random numbers used as destination addresses to be put in the header of the input test packets to the router. It is comprised of a random generator state machine that generates a wide enough random register (12-bit) to spread the widen circulation of the generator (the interval in which the generator generates its random pattern and starts to give back its random pattern). Only seven bits are used as a random destination address. Because the generator state machine never generates all zeros, so the seven bits are chosen to be bit 7 down to 1. Then, a zero is appended to the most significant bit to form eight bits. The eight bits are then converted from serial to parallel vector of 128 random addresses. Figure 4.1 shows the process of generating the random pattern and putting the seven bits destination address in the header of the packet.

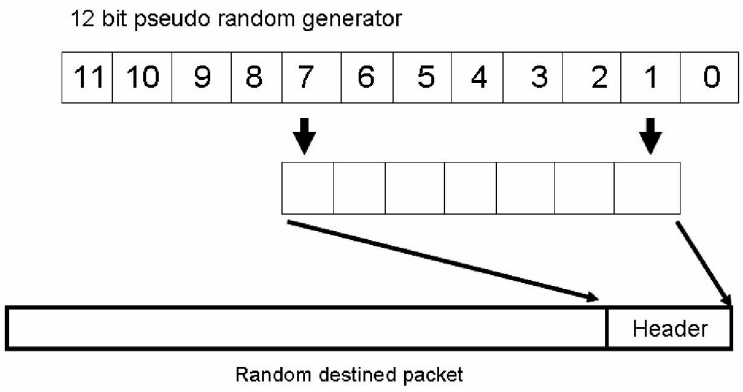


Figure 4.1: Generating the random destination address.

4.1.3 Time Counter

A four bytes counter is used to stamp the packet with the time of its egress from the load generator and ingress to the router. The counter counts every router clock *sw_clk*. Figure 4.2 shows the time stamping process of the packet.

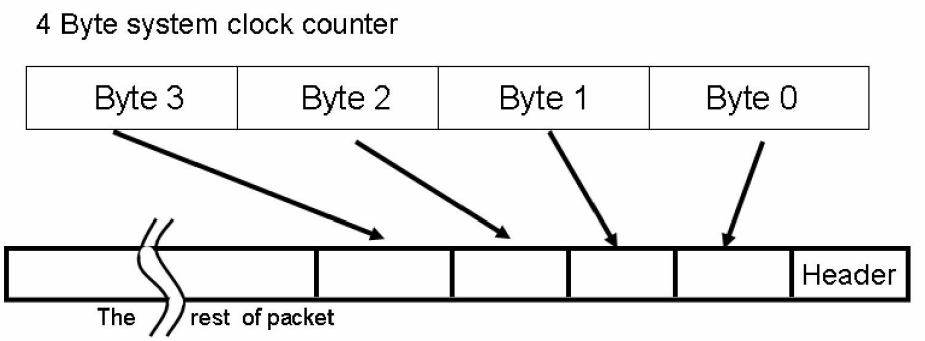


Figure 4.2: The time stamping of the packets.

4.1.4 Traffic Load Adjust Counter

The traffic load adjust counter is simply a counter mainly used to count the packet width (the number of phits in the packet) and other cycles can be added as a dummy interval between the packets to reduce the traffic load. If the counter counts just the packet width, the traffic load will be 100%. If dummy cycles are added, the traffic load will be reduced by the ratio of the inserted cycles and the width of the packet. That is used to vary the load input to the router to let the packet counter calculate the router average delay at various traffic loads. The traffic load adjust counter state machine counter counts each clock cycle.

4.1.5 The Main Controller

The main controller is responsible for assigning both the random destination address and the time stamp to the packets and adjusting the input traffic load to the router. It can also control the time of exposing the router to the specified load by changing the amount of packets entering the router with the specified traffic load during simulation. The load generator is a state machine consisting of five states.

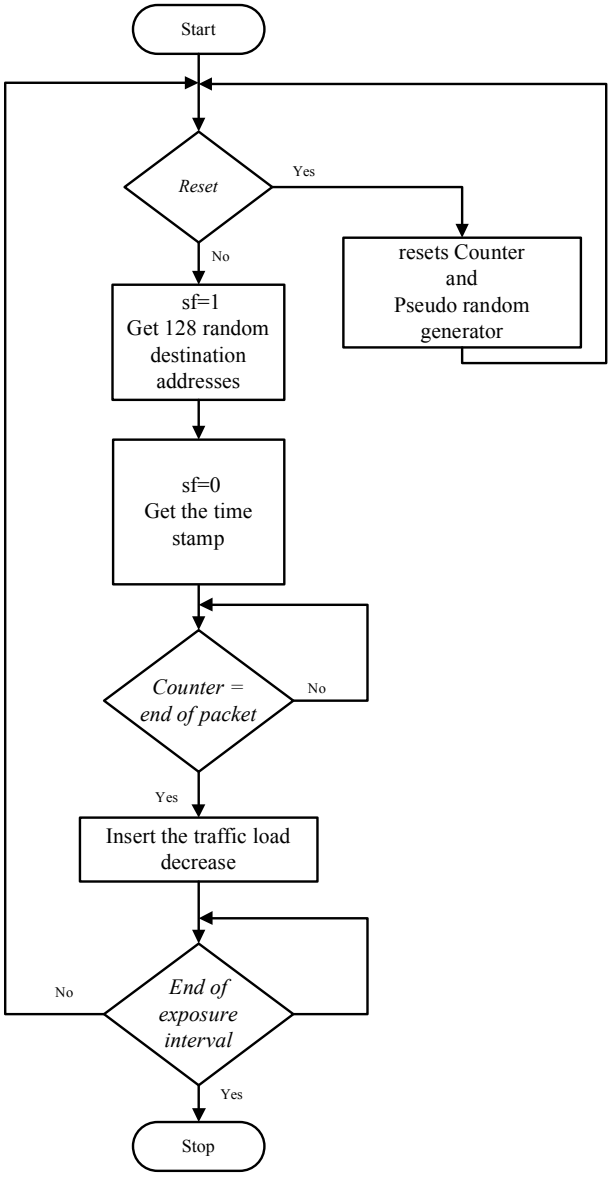


Figure 4.3: Flow chart of the load generator.

At reset, the load generator resets the traffic load counter and stops the Pseudo random generator state machine. At state zero, all the 128 start of packets (*sf*'s) associated to all generated packets are set to one. The destination addresses are let out as the first phits of the 128 packets.

At state one through four, the time stamp is assigned to the output packets. At state five, the rest of the packet plus the traffic load decrease is counted. Then, the number of packets is counted to stop working after a specified exposure

interval. Figure 4.3 shows the flow chart of the load generator. Figure 4.4 shows the block diagram of the load generator with respect to the rest of the components.

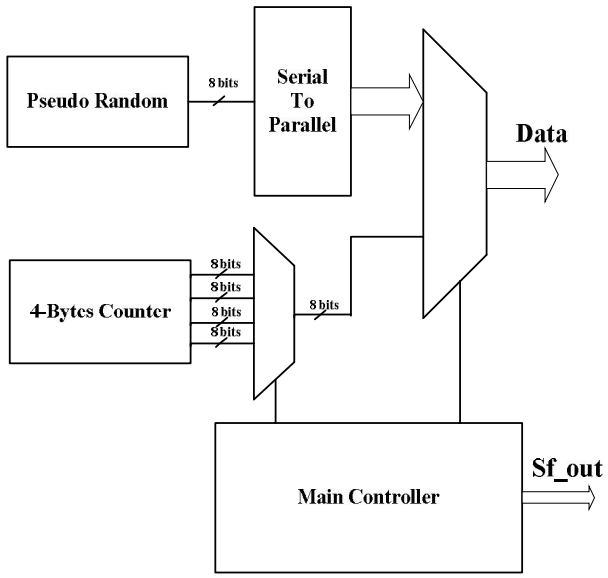


Figure 4.4: Load generator with other components.

4.2 The Packet Counter

The Packet Counter is used to calculate the throughput and the average delay. It consists of two components; the packet counter, the average delay calculator, and summation. All components are discussed in the following subsections.

4.2.1 Packet Calculator

There is a packet calculator on each output port of the router. The packet calculator is responsible for counting the number of output packets from the router to calculate the throughput.

4.2.2 Average Delay Calculator

The average delay calculator subtracts the time stamp assigned to each packet from the current time of the time counter to calculate the time difference. Then the time difference is accumulated for all the output packets.

4.2.3 Summation

Summation is used to calculate the total output packets by summing all the accumulated number of output packets. The total number of time differences is also calculated by summing all the accumulated time differences from the entire time calculator state machines to get the total time differences.

The throughput is calculated manually from ModelSim by dividing the total output packets by the number of packets exposed to the router in the exposure time.

The average delay is calculated manually by dividing the summation of time differences by the number of output packets. Because the internal buffers are nearly empty at 20% and 80% of the exposure time, so the average delay must not be taken into consideration under 20% and above 80% exposure time. So, the average delay is calculated only between 20% and 80% of the exposure time. Figure 4.5 shows the router with respect to the simulation environment.

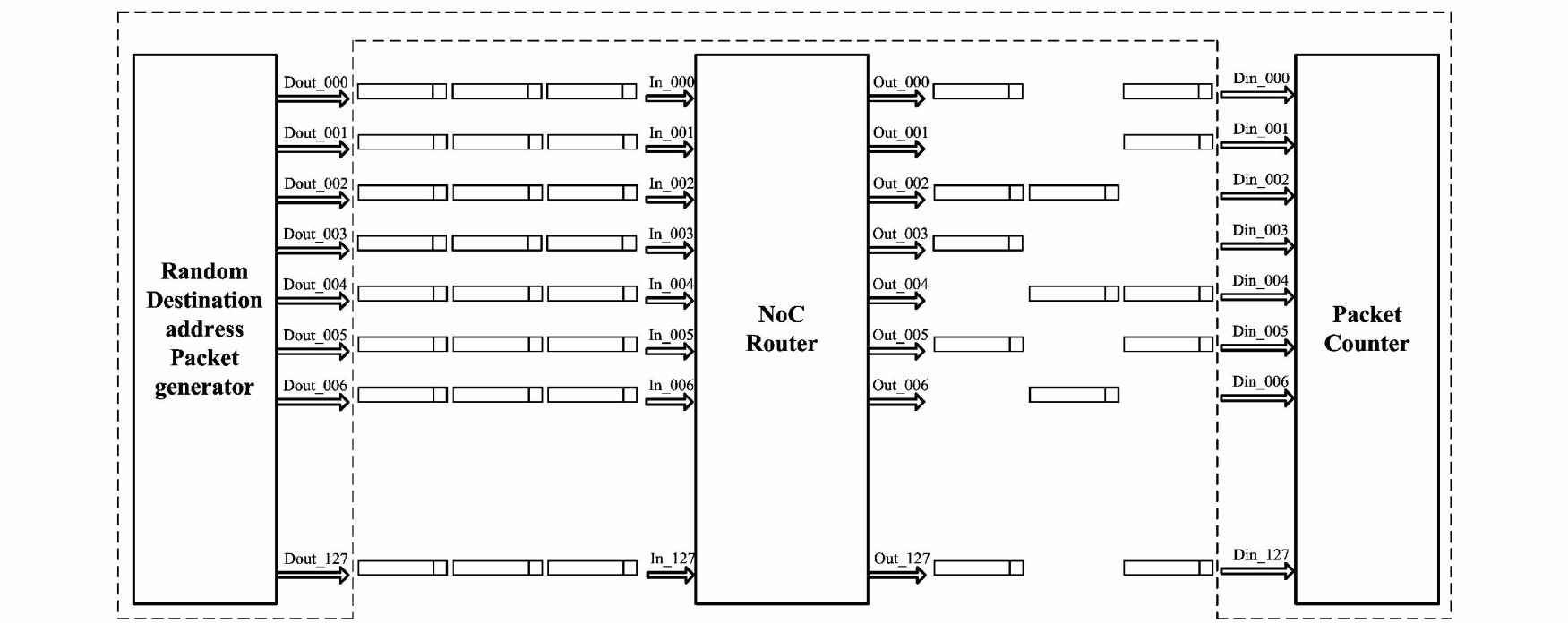


Figure 4.5: The simulation environment including the router.

This Page Intentionally Left Blank

Chapter 5

Evaluation Results

The intended functionality and the associated software applications executed on the NoC determine the amount of traffic load between nodes and hence the performance required by the used router. The router performance is affected by the amount of buffers and the buffering strategy. More buffer space gives higher performance. We have run our simulations with a range of different workloads and for different input buffer sizes.

5.1 Results for 128×128 Switch/Mesh Network

In FIFO input buffering strategy, throughput is limited due to (HOL). The throughput is tested with varying the traffic load for 32, 64, and 128 blocks input buffer for time duration of 1000 packets input to each port. As expected, larger buffers enhance the performance but it is limited by the maximum throughput in input buffering system due to HOL blocking in line with the results of [37]. Figure 5.1 shows the FIFO throughput with traffic load. The maximum achieved throughput was 50% at 50% traffic load for 128 packets buffer. Throughput can approach the theoretical maximum (58.6%) by increasing the buffer space.

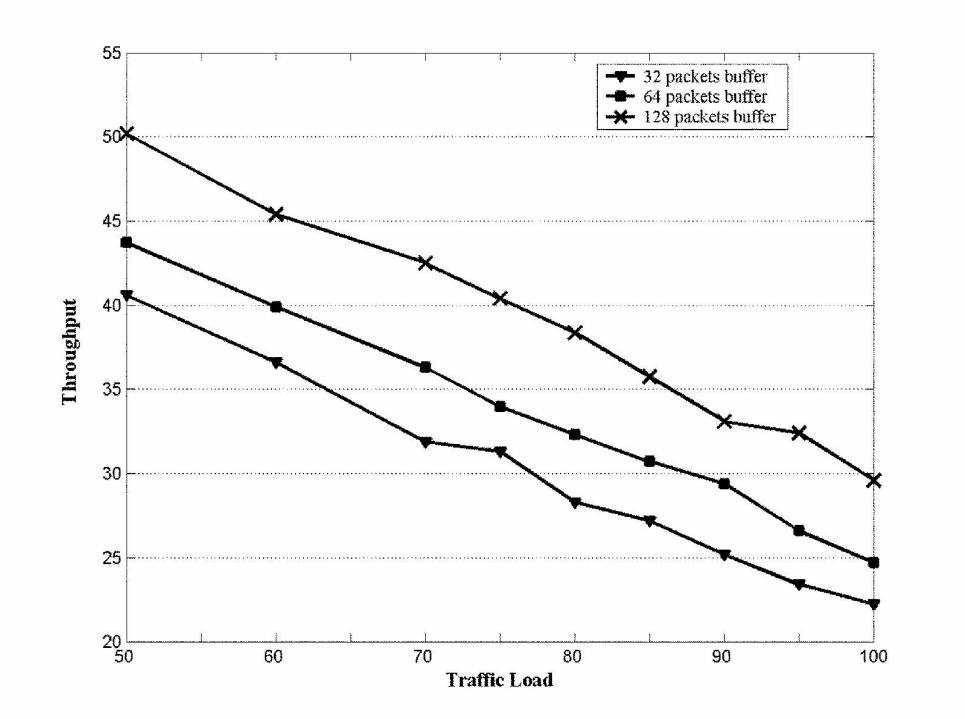


Figure 5.1: The average latency of the FIFO router with the exposed load.

VOQ buffering strategy gives optimum performance (100% throughput) when exposed to 97% traffic load. The throughput is tested for a time duration of 100,000 input packets on each port. For 100% traffic load, optimum performance can be obtained by using infinite buffer size. For finite buffer sizes, throughput will start to decrease (packets will start to be dropped) at certain exposure duration (time of a number of packets input to each port as test). Depending on the amount of buffers, packets are dropped due to the lack of buffer space. Increasing the size of the switch (number of input/output) will have a slight effect on the throughout at 100% traffic load whereas throughput is 100% under lower loads. Figure 5.2 shows the throughput of the router with the exposure time duration under 100% traffic load.

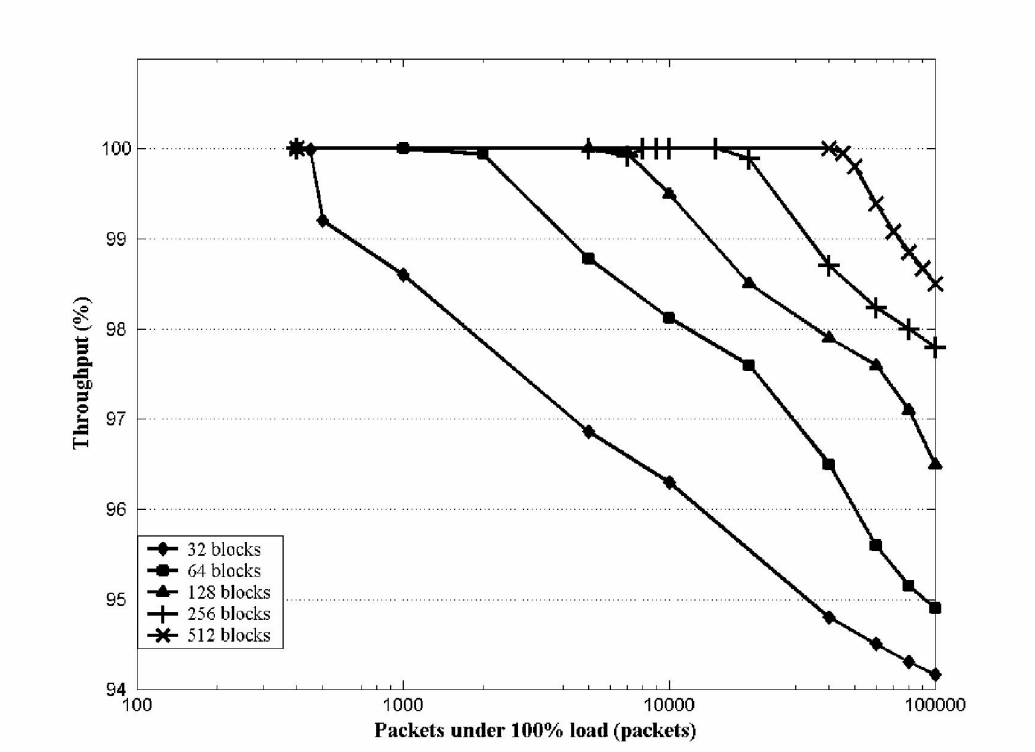


Figure 5.2: The throughput of the VOQ router with the exposure time duration under 100% traffic load.

Router latency is the average delay required to transfer a packet from the input to the output through the router. Router latency comes from the contention between the arriving packets in addition to the internal logic and buffering delay. Increasing the size of the switch will slightly affect the latency due the larger logic. Latency is affected mainly by the load on the router. As load increases, packet average latency increases. At high traffic loads, latency increases exponentially. Figure 5.3 shows the measured average latency in arbitration cycles with different traffic loads. The figure shows also the standard deviation of the average latency. If bus width of the router is wide enough to hold the complete needed packet (no serialization), the arbitration cycle will be one system clock cycle.

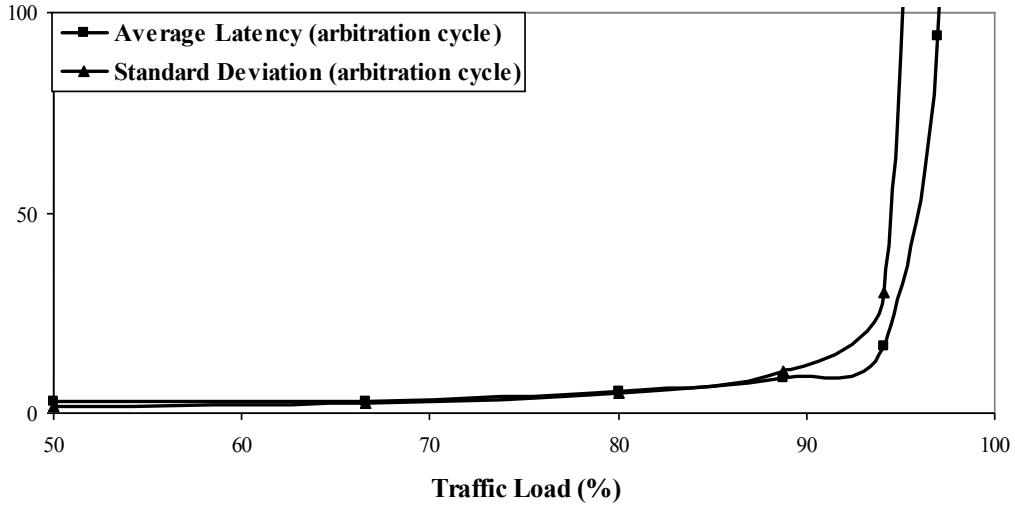


Figure 5.3: The average latency and its standard deviation for the VOQ router with the exposed load.

The designs are synthesized using Synopsys Design Compiler using TSMC 65 nm standard cell library. First, both Crossbar and Batcher-Banyan 128×128 switching core alone are synthesized. The Crossbar occupies 147% larger area because it uses more crosspoints than the Batcher-Banyan. However, the Batcher-Banyan consumes 120% more dynamic power consumption because each Batcher-Banyan node contains sequential control logic for the self routing property which consumes more power. Figure 5.4 shows both area and dynamic power consumption of synthesized gates of the switching cores.

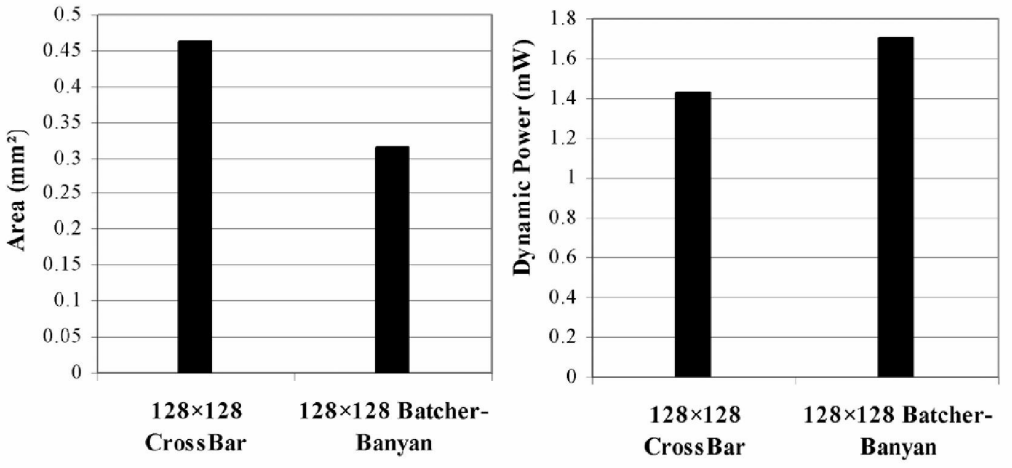


Figure 5.4: Area and power of Batcher-Banyan vs. crossbar switching cores.

For area costs of the router architectures (excluding the buffer memory), FIFO input with Batcher-Banyan shows the least area because the simple logic of all of the FIFO input and the Ring-reservation arbitration unit. Both the VOQ's occupy about five times larger area than the FIFO. Whereas the sum of 128 small 5×5 VOQ-Crossbar has about three times larger area than the FIFO. The VOQ-Crossbar occupies only 1.04 times the VOQ-Batcher-Banyan area because the arbiter dominates the area over the switching cores where the area of the DPM arbiter increases exponentially with the size of the router. Figure 5.5 shows the area costs of the router architectures excluding the buffer memory.

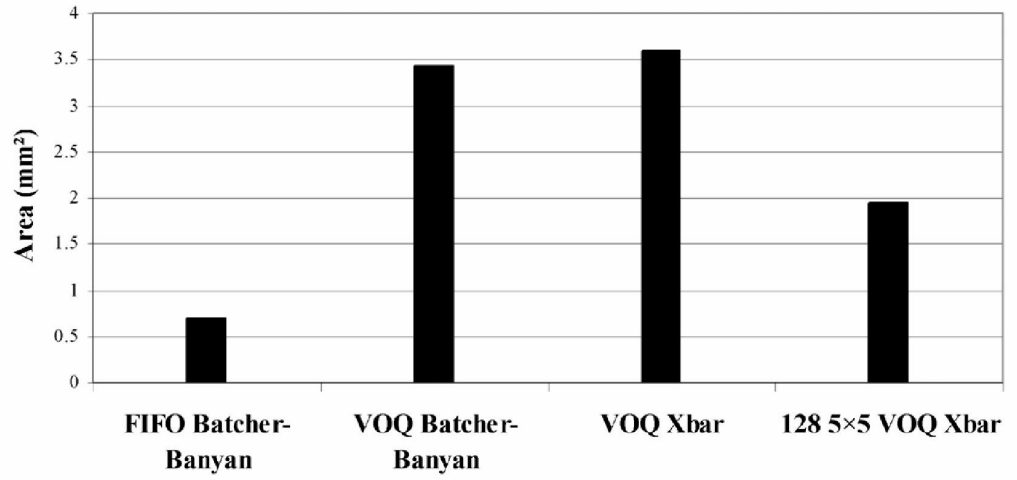


Figure 5.5: Area costs of the full router architectures excluding the buffer memory

For the dynamic power consumptions of the router architectures (excluding the buffer memory), FIFO input consumes very low power compared with the VOQ's because of its simple logic. Whereas, the sum of 128 small 5×5 VOQ-Crossbar consumes about three times the FIFO power. The VOQ-Crossbar showed different results than expected from the previous results of the switching cores power consumptions. The previous results of the switching cores show that the Batcher-Banyan switching core consumes more power than the crossbar. The power consumptions of the full router architectures show different results. The results show that the VOQ-Crossbar router consumes more power than the VOQ-Batcher-Banyan by 1.03. The increase in the VOQ-Crossbar router comes from the control signals (wires) needed to control (turn on and off) the crosspoints. These wires connect the crosspoits with the arbitration unit, and that causes more power consumption in the internal wires. Whereas the Batcher-Banyan has a self-routing property and does not need any control from outside. The power consumption of the large VOQ's is about ten times that of 128 small 5×5 VOQ-Crossbar because all of the VOQ logic, DPA and the Crossbar exponential growth in complexity with the increase in the number of ports. Figure 5.6 shows the dynamic power consumption of the router architectures.

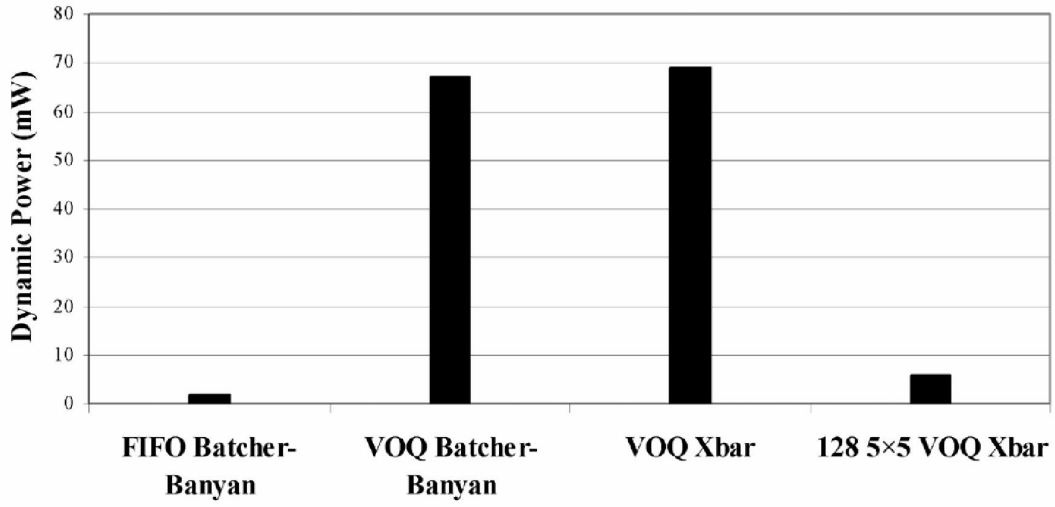


Figure 5.6: Dynamic power consumption of the full router architectures excluding the buffer memory

Area and power efficiencies are calculated for the designs. Area efficiency is the throughput to the unit area, and it is calculated by dividing the throughput by the area. Also, power efficiency is the throughput to the unit power, and it is calculated by dividing the throughput by the power. The FIFO input router shows the highest efficiency despite its low throughput because its very low area and power consumption. The efficiency of 128 small 5×5 VOQ-Crossbar is in the second stage after the FIFO. Both the 128×128 VOQ gives lower area efficiency and a lot lower power efficiency. VOQ-Batcher-Banyan gives slightly more area and power efficiencies than the VOQ-Crossbar. Figure 5.7 shows area efficiency and Figure 5.8 shows power efficiency of the full routers.

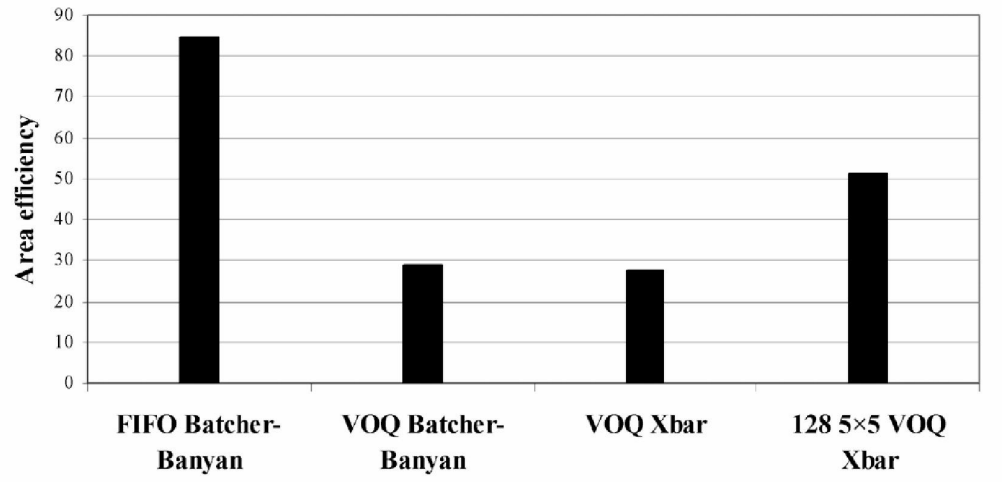


Figure 5.7: Area efficiency of the full routers.

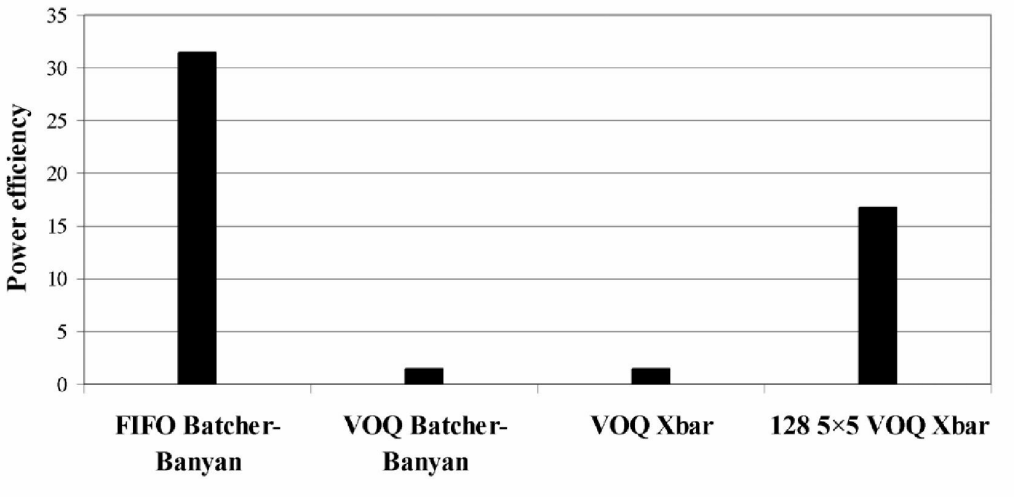


Figure 5.8: Power efficiency of the full routers.

The results showed that the FIFO with Batcher-Banyan is the highest efficiency because of the simple logic of the FIFO input units and its associated arbitration unit. The 128 5×5 VOQ-Crossbar comes in the second stage. Both the VOQ with Batcher-Banyan and Crossbar showed the lowest efficiency because the complex logic of the VOQ input units and its associated arbitration. VOQ with Batcher-Banyan shows higher area efficiency and lower power efficiency than VOQ with Crossbar.

5.2 Predictions and Extrapolation for Larger Switch/Mesh Network Sizes (256×256 and above)

The average delay slightly increases for the large switches due the larger logic, buffering, and witching nodes. But in case of larger mesh network utilizing 5×5 routers, average delay will increase linearly due to the increase in the diameter of the mesh network (network diameter is largest, minimal hop count over all pairs of terminal nodes in the network). The FIFO Ring Reservation arbitration cycle grows linearly with the switch size but this effect can be cancelled by increasing the packet flits to greater than or equal the switch size N.

In mesh networks, the total area grows linearly with the number of nodes. In other words, doubling the network size will double the area occupied with the communication routers.

For FIFO switches, area can be divided into four parts. Input units grow linearly with the switch size N. The Ring Reservation arbiter grows in order of N CSI's where there is always one RHE unit. The Batcher network grows as $N/2\sum_{i=1}^{i=n} i$, where $n = \log_2 N$. The Banyan network grows as $N/2\log_2 N$.

For the VOQ with Batcher-Banyan, area can be divided into four parts. Input units grow linearly with the switch size N. The DPA grows as 2N(N-1). The Batcher network grows as $N/2\sum_{i=1}^{i=n} i$, Where $n = \log_2 N$. The Banyan network grows as $N/2Log_2N$.

For The VOQ with Crossbar, area can be divided into four parts. Input units grow linearly with the switch size N. The DPA grows as 2N(N-1). The Crossbar grows as N². Figure 5.9 shows the growth of area of the four switches with the switch/network sizes from 32×32 to 128×128. The figure shows that the area of the VOQ with Batcher-Banyan starts to be effectively lower than the VOQ with Crossbar starting from 128×128 switch size.

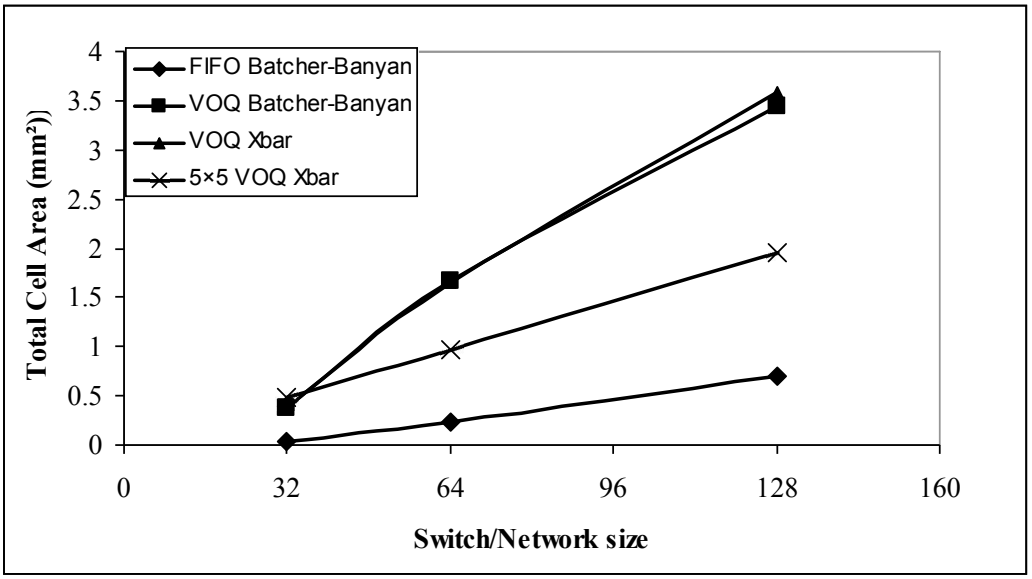


Figure 5.9: Area growth vs. router sizes from 32×32 to 128×128 input/outputs.

Figure 5.10 Shows the growth of power consumption of the four switches with the switch/network sizes from 32×32 to 128×128. The figure shows that the total dynamic power of the VOQ with Batcher-Banyan starts to be effectively lower than the VOQ with Crossbar starting from 128×128 switch size. The following table summarizes the previous results.

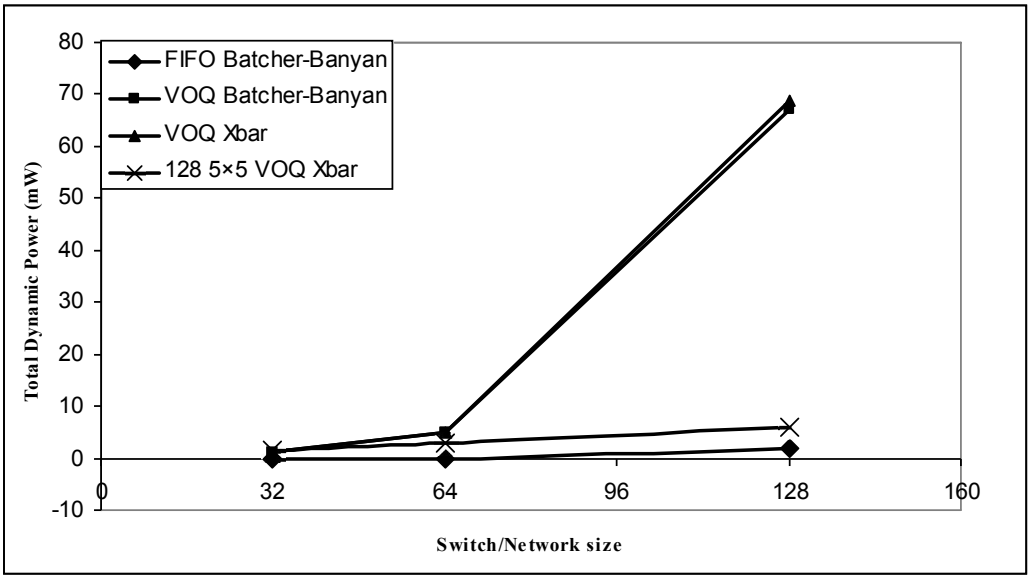


Figure 5.10: Total dynamic power growth vs. router sizes from 32×32 to 128×128 input/output.

	Mesh Network	FIFO	VOQ-Batcher-Banyan	VOQ-Crossbar
Latency	O(N)	Slight increase	Slight increase	Slight increase
Area	O(N)	$N + N + N/2\sum_{i=1}^{i=n} i + N/2\,Log_2 N$, Where $n = \log_2 N$	$N + 2N(N-1) + N/2\sum_{i=1}^{i=n} i + N/2\,Log_2 N$, Where $n = \log_2 N$	$N + 2N(N-1) + N^2$

This Page Intentionally Left Blank

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This work discussed the future many core NoC architectures employed by star, hierarchical-star, and fat-tree network topologies and large size routers. Three 128×128 router designs are implemented, the first is FIFO input with Batcher-Banyan switching core, the second is VOQ input with Batcher-Banyan and the last is VOQ input with Crossbar switching core. The three router designs are evaluated according to throughput, area, and power. A network simulation environment is developed to test the large routers for throughput and average delay under various loads and number of input packets. The efficiencies of the three designs are evaluated compared to 128 5×5 VOQ-Crossbar routers employed in conventional 2-D mesh. The results showed that the FIFO with Batcher-Banyan yields the highest efficiency in area and power consumption followed by the 128 5×5 VOQ-Crossbar. Both the VOQ with Batcher-Banyan and Crossbar showed the lowest efficiency. However, the FIFO technique is limited in its maximum throughput due to its maximum throughput due to the HOL problem. For higher throughput, any VOQ router is recommended. The average delay of the distributed 5×5 VOQ-Crossbars in mesh network is very high where the packets have to cross many routers through their way from source to destination.

At scaling up the switch/network size (above 128), VOQ with Batcher-Banyan will have lower area and power than the VOQ with Crossbar. Area and power of the other types grow linearly.

One large VOQ router in star topology for example shows better average latency than distributed 128 5×5 routers in mesh topology due to the

accumulated delay in the hops that packet has to pass to reach its destination. The problem of long arbitration cycle of the FIFO Ring Reservation can be eliminated by using long packet size.

For designs where throughput is not a problem, FIFO is the best choice for its low area and power consumption. If throughput is a real concern, a compromise will be upon average latency, area, and power. If average latency is not a concern, distributed 5×5 routers in mesh topology NoC will be the best choice. If average delay is a concern, a compromise between VOQ with Batcher-Banyan and VOQ with Crossbar according to their area and power consumption.

6.2 Summary of Contributions

This work has put a proposal to the architecture of future many-core NoC where the cores will be heterogeneous their numbers will go beyond the thousands. In this work, three architectures of large routers are discussed in detailed. Comparisons between these three architectures are stated with respect to throughput, area, power, and performance.

6.3 Future Work

In the future we or other teams can enhance the switch functionality, maybe by increasing the number of input/output, adding error checking bits, adding asynchronous transmission, routing algorithms and multicast, etc.

References

[1]. Intel, "Intel® Core™2 Duo Processor," http://www.intel.com/products/processor/core2duo/index.htm, 2012.

[2]. J. Held, J. Bautista,S. Koehl. "Overview, From a Few Cores to Many: A Tera-scale Computing Research." Intel. 2006. ftp://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf.

[3]. S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS", IEEE Journal of Solid State Circuits 43.1 (2008) : 29-41.

[4]. IBM, "Cell Broadband Engine Architecture and its first implementation," http://www.ibm.com/developerworks/power/library/pa-cellperf/, 2012.

[5]. Sun microsystems, "MAJC™ ARCHITECTURE TUTORIAL," http://java.sun.com/images/tutorial.pdf, 2012.

[6]. Oracle, "SPARC T4 PROCESSOR," http://www.oracle.com/us/products/servers-storage/servers/sparc-enterprise/t-series/sparc-t4-processor-ds-497205.pdf, 2012.

[7]. Ambric, "Am2045," http://www.ambric.com/products_mppa.php, 2012.

[8]. Nvidia, "TESLA™ C2050 / C2070 GPU COMPUTING PROCESSOR SUPERCOMPUTING AT 1/10Th ThE COST," http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_C2050_C2070_jul10_lores.pdf, 2012.

[9]. R. Chan-Eun, J. Han-You and S. Ha, "Many-to-Many Core-Switch Mapping in 2-D Mesh NoC Architectures", Proc of The 22nd IEEE International Conference on Computer Design (ICCD 2004), San Jose, CA, USA, 11-13 October 2004, pp. 438–443.

[10]. S. Mubeen, "Evaluation of source routing for mesh topology network on chip platforms", master of science thesis at Jönköping Institute of Technology within the subject area Electronics, 2009.

[11]. J. Held, J. Bautista and S. Koehl. "From a Few Cores to Many: A Tera-Scale Computing Research Overview", White Paper research at Intel, 2006.

[12]. L. Benini and G. D. Micheli, “Powering Networks-on-Chips: Energy-efficient and Reliable Interconnect Design for SoCs”, Proceedings of the 14th International Symposium on Systems Synthesis (ISSS 2001), Montréal, Québec, Canada, September 30 - October 3, 2001 pp. 33-38.

[13]. International Technology Roadmap for Semiconductors, http://www.itrs.net/Links/2003ITRS/Home2003.htm, 2012.

[14]. C.S. Bamji, M. Berkens, A. B. Kahng and C. Strolenberg, “Automated Layout and. Migration in Ultra-Deep Submicron VLSI”, In Proc. 36th Design Automation Conference, New Orleans, LA, USA, 21-25 June 1999.

[15]. L. Benini, G. D. Micheli, “Networks-on-Chips: a new SoC paradigm”, IEEE Computer, vol. 35, no. 1, pp. 70-78, Jan 2002.

[16]. Y. Pan, P. Kumar, J. Kim, G. Memik, Y. Zhang, A. Choudhary, “Firefly: illuminating future network-on-chip with nanophotonics”, ACM SIGARCH Computer Architecture News 37.3 (December 2009) : 429-440.

[17]. L. Benini, and G. D. Micheli, “Networks-on-Chips: Technology and Tools”, Morgan Kaufmann 2006.

[18]. O. Villa, G. Palermo and C. Silvano, “Efficiency and Scalability of Barrier Synchronization on NoC Based Many-core Architectures”, In Proceedings of the 2008 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2008), Atlanta, GA, USA, October 19-24, 2008, pp. 81-90.

[19]. Tilera, “TILE-Gx processors family,” http://www.tilera.com/products/processors/TILE-Gx_Family, 2012.

[20]. W. J. Dally and B. Towles, “Route packets, not wires: on-chip interconnection networks”, In Proc. 38’th Design Automation Conference (DAC'01), June 18-22, 2001, Las Vegas, USA.

[21]. H. Wang, L.-S. Peh, and S. Malik, “A technology-aware and energy oriented topology exploration for on-chip networks”, In Proceeding of Design, Automation and Test in Europe Conference (DATE 2005), Munich, Germany, March 7-11, 2005, 1238-1243.

[22]. J. S. Kim, M. B. Taylor, J. Miller, and D. Wentzlaff, “Energy characterization of a tiled architecture processor with on-chip networks”, In Proceeding of International Symposium on Low Power Electronics and Design (ISLPED 2003), Seoul, Korea, August 25-27, 2003, pages 424–427.

[23]. T. T. Ye, L. Benini, G. D. Micheli, "Analysis of Power Consumption on Switch Fabrics in Network routers", In Proceeding of the 39th Design Automation Conference (DAC 2002), New Orleans, LA, USA, June 10-14, 2002.

[24]. C. Xuning and L. S. Peh, "Leakage power modeling and optimization in interconnection networks", In Proceeding of the International Symposium on Low Power Electronics and Design (ISLPED 2003), Seoul, Korea, August 25-27, 2003, pp. 90–95.

[25]. H. Wang, L. Peh, and Sharad Malik "Power-Driven Design of Router Microarchitectures in On-Chip Networks", In Proceedings of the 36th Annual International Symposium on Microarchitecture (MICRO 2003), San Diego, CA , USA, December 3-5, 2003.

[26]. K. Lee, S.-J. Lee, and H.-J. Yoo, "Low-power network-on-chip for high-performance SoC design", IEEE Transactions on VLSI Systems, 14(2): 148-160 (2006)

[27]. S. Lee, K. Lee, H. Yoo, "Analysis and Implementation of Practical, Cost-Effective Networks-on-Chips", IEEE Design and Test of Computers, Vol.22, No.5, pp. 422-433.

[28]. C. D. Thompson, "A Complexity Theory for VLSI, PhD thesis", Carnegie-Mellon University, August 1980.

[29]. M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, "The Raw microprocessor: A computational fabric for software circuits and general-purpose programs", In Proceeding of the 35th International Symposium on Microarchitecture (MICRO 2002), Istanbul, Turkey, November 18-22, 2002, 22(2):25–35.

[30]. H. Wang, X. Zhu, L. Peh, and S. Malik, "Orion: A power-performance simulator for interconnection networks", In Proc. 35th International Symposium on Microarchitecture (MICRO 2002), Istanbul, Turkey, November 18-22, 2002, pages 294–305.

[31]. J. Chan and S. Parameswaran, "NoCEE: Energy macro-model extraction methodology for network on chip routers", In Proceeding of the International Conference on Computer-Aided Design (ICCAD 2005), San Jose, CA, USA, November 10-13, 2005, pp. 254-259.

[32]. A. Bogliolo, L. Benini, and G. D. Micheli, "Regression-based RTL power modeling", ACM Transaction on Design Automation of Electronic systems, 372–337 ,(3)5, April 2000.

[33]. S. Penolazzi, "An empirical power model of the links and the deflective routing switch in nostrum", Master's thesis, School for Information and Communication Technology, Royal Institute of Technology, Stockholm, Sweden, December 2005.

[34]. E. Nilsson, "Design and implementation of a hot-potato switch in a network on chip", Master's thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, IMIT/LECS 2002-11, Stockholm, Sweden, June 2002.

[35]. G. Passas, M. Katevenis, and D. Pnevmatikatos, "A 128 × 128 × 24Gb/s crossbar interconnecting 128 tiles in a single hop and occupying 6% of their area", In Proceeding of the 4th ACM/IEEE International Symposium on Networks-on-Chip (NOCS), Grenoble, France, May 3-6, 2010, (pp. 87-95) IEEE Computer Society.

[36]. W. J. Dally and B. Towles, "Principles and Practices of Interconnection Networks", Morgan Kaufmann, 2004

[37]. M. J. Karol, M. Hluchyj, and S. Morgan, "Input vs. output queuing on a space-division packet switch", in Proceeding of the Global Communications Conference, (GLOBECOM 1986), Houston, USA, December 2-4, 1986, pp. 659–665.

[38]. Y. Tamir and G. L. Frazier, "High-Performance Multi-Queue Buffers for VLSI Communication Switches", In Proceeding of the 15th Annual International Symposium on Computer Architecture (ISCA 1988), Honolulu, Hawaii, USA, May-June 1988, pp. 343-354.

[39]. N. Ni, M. Pirvu, and L. Bhuyan, "Circular buffered switch design with wormhole routing and virtual channels", In Proceeding Of the International Conference on Computer-Aided Design (ICCAD 1998), San Jose, CA, USA, November 1998.

[40]. M. Fayyazi, D.R. Kaeli, and Z. Navabi, “Dynamic Input Buffer Allocation (DIBA) for Fault Tolerant Ethernet Packet Switching”, in Proceeding of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2003), Las Vegas, Nevada, USA, June 23-26, 2003, pp. 819-823.

[41]. L. R. Goke and G. J. Lipovski, “Banyan networks for partitioning multiprocessor systems”, In Proceeding of the 1st Annual International Symposium on Computer Architecture (ISCA 1973), Gainesville, FL, USA, December 9-11, 1973, pp. 21-28.

[42]. H. J. Chao, C. H. Lam, E. Oki, “Broadband Packet Switching Technologies: A Practical Guide to ATM Switches and IP Routers”, Wiley-Interscience 2001.

[43]. K. E. Batcher, “Sorting networks and their application”, In Proceeding of the Spring Joint Computer Conference (AFIPS 1968), Atlantic City, NJ, USA, 30 April - 2 May 1968, pp.307-314.

[44]. J. Hurt, A. May, X. Zhu, and B. Lin, “Design and implementation of high-speed symmetric crossbar schedulers”, In Proceeding of the IEEE International Conference on Communications (ICC’99), Vancouver, Canada, June 5-9, 1999, pp. 253-258.

[45]. N. McKeown, V. Anamtharam, and J. Warland, “Achieving 100% throughput in an input-queued switch”, In Proceeding of the INFOCOM’96, , The Conference on Computer Communications, Fifteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Networking the Next Generation, San Francisco, March 1996, pp. 296-302.

[46]. B. Bingham and H. Bussey, “Reservation-based contention resolution mechanism for Batcher-banyan packet switches”, Electron. Lett., vol. 24, no. 13, pp. 772 773, 23 Jun. 1988.

[47]. Y. Tamir, H. C. Chi, “Symmetric crossbar arbiters for VLSI communication switches”, IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 1, pp. 13-27, 13–27 January 1993.

[48]. K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, “Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture”, In Proceeding of the International Symposium on Computer Architecture (ISCA 2003), San Diego, California, USA, 9-11 June 2003, pages 422–433.

[49]. F. Angiolini, P. Meloni, S. Carta, L. Benini and L. Raffo, “Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness”, Proceedings of the Design, Automation and Test in Europe Conference (DATE 2006), Munich, Germany, March 6-10, 2006, Vol. 1, pp.36.

[50]. T. Bjerregaard and S. Mahadevan, “A survey of research and practices of network-on-chip”, ACM Computing Surveys, 38(1), 2006.

Appendix: Practical Operation Examples

A. Linked List Control

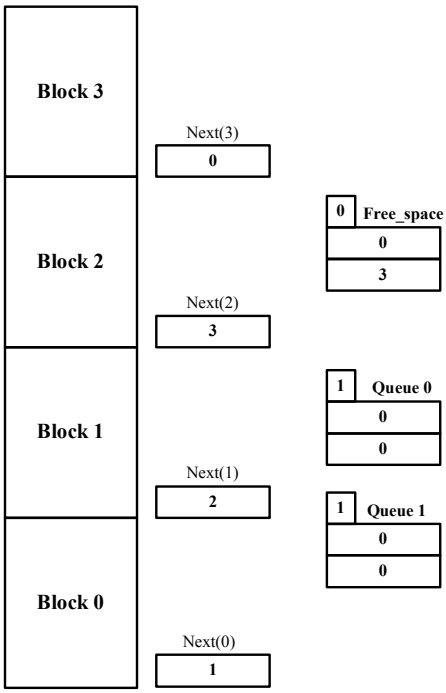
Figure A.1 shows data changes in the linked list registers of a buffer consisting of four blocks and two virtual queues. The example traces data when queuing and dequeuing packets to and from the buffer. The figure contains sub figures showing the buffer and the values of the linked list.

Figure A.1 (a) shows the reset state of the registers where each block's next register points to the next block and both queues are empty and the free space buffer is not empty and all heads and tails point to block zero.

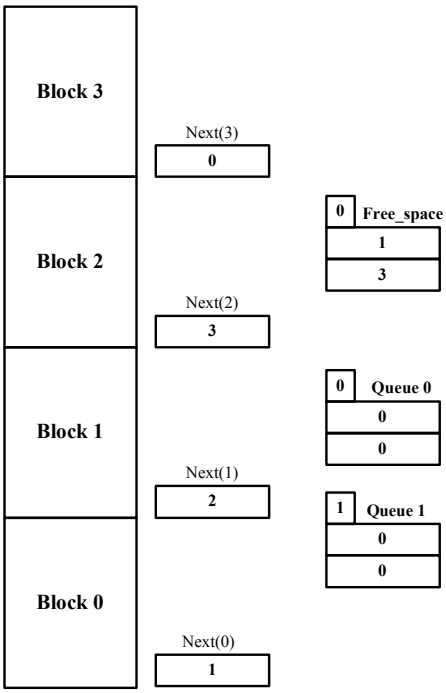
Figure A.1 (b) shows the state after receiving a packet addressed to queue zero. The packet is put to the head of the free space queue (block zero) then moved to be the head and tail of queue zero which was empty. So, queue zero became not empty after it was empty. Its empty flag is reset to zero and its head and tail is now pointing to block zero. Free space head now points to block one.

Figure A.1 (c) shows the state after receiving a packet addressed to queue one. The packet is put to the head of the free space queue (block one) then moved to be the head and tail of queue one which was empty. So, queue one became not empty after it was empty. Its empty flag is reset to zero and its head and tail is now pointing to block one. Free space head now points to block two.

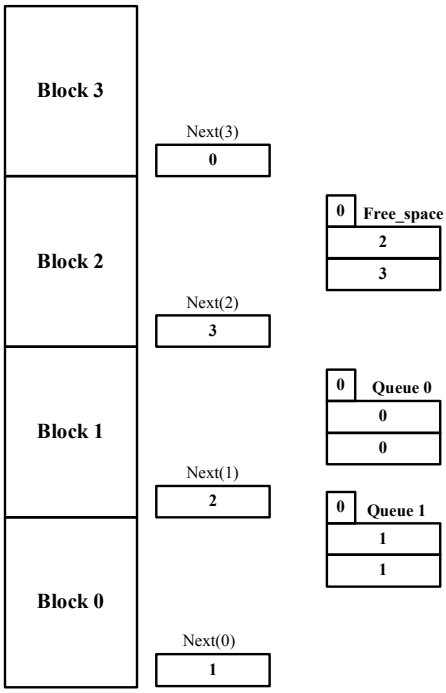
Figure A.1 (d) shows the state after receiving another packet addressed to queue zero. The packet is put to the head of the free space queue (block two) then moved to be the tail of queue zero. Its tail is now pointing to block two. The next register of the old tail (block zero) is now pointing to the new tail (block two). Free space head now points to block three.


Block 3
Block 2
Block 1
Block 0
Next(3)
0
Next(2)
3
Next(1)
2
Next(0)
1
0 Free_space
0
3
1 Queue 0
0
0
1 Queue 1
0
0

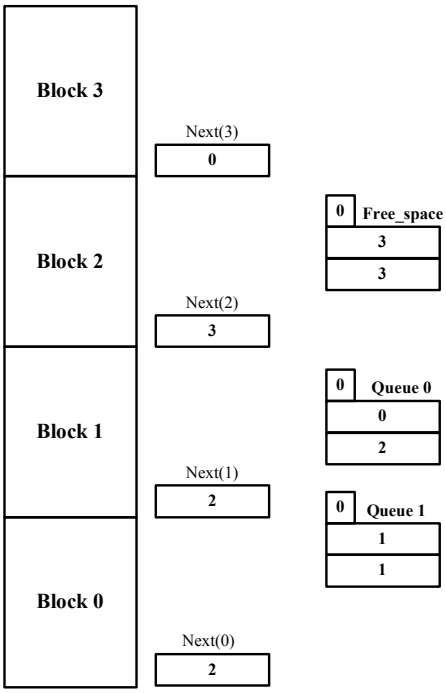
(a)


Block 3
Block 2
Block 1
Block 0
Next(3)
0
Next(2)
3
Next(1)
2
Next(0)
1
0 Free_space
1
3
0 Queue 0
0
0
1 Queue 1
0
0

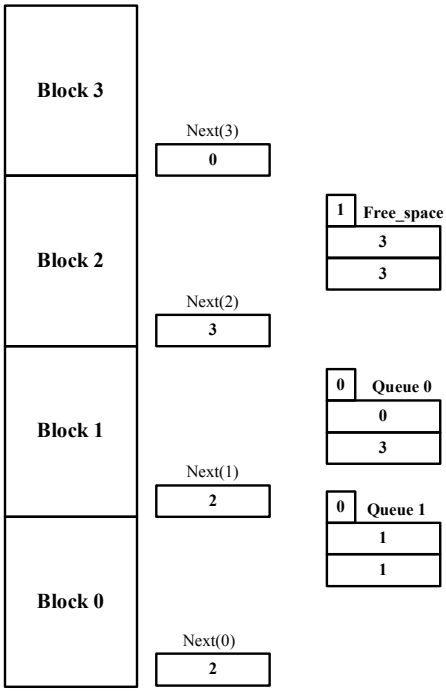
(b)


Block 3
Block 2
Block 1
Block 0
Next(3)
0
Next(2)
3
Next(1)
2
Next(0)
1
0 Free_space
2
3
0 Queue 0
0
0
0 Queue 1
1
1

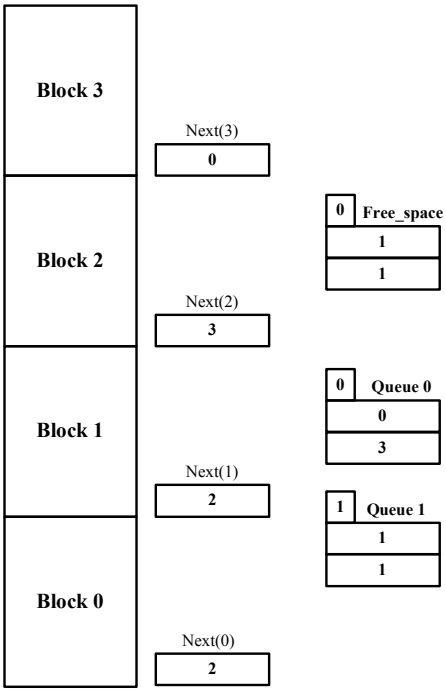
(c)


Block 3
Block 2
Block 1
Block 0
Next(3)
0
Next(2)
3
Next(1)
2
Next(0)
2
0 Free_space
3
3
0 Queue 0
0
2
0 Queue 1
1
1

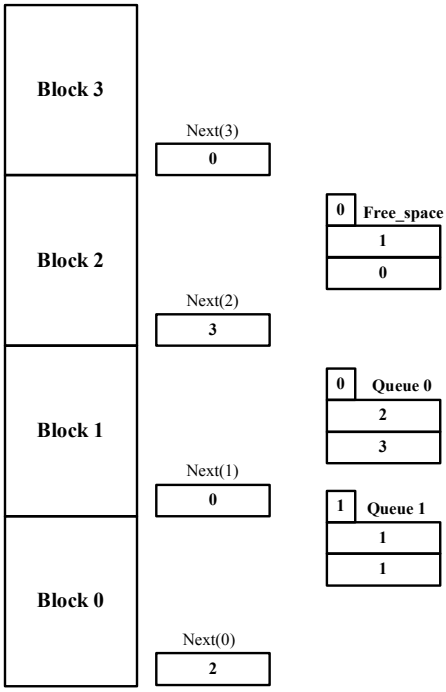
(d)


Block 3
Block 2
Block 1
Block 0
Next(3)
0
Next(2)
3
Next(1)
2
Next(0)
2
1 Free_space
3
3
0 Queue 0
0
3
0 Queue 1
1
1

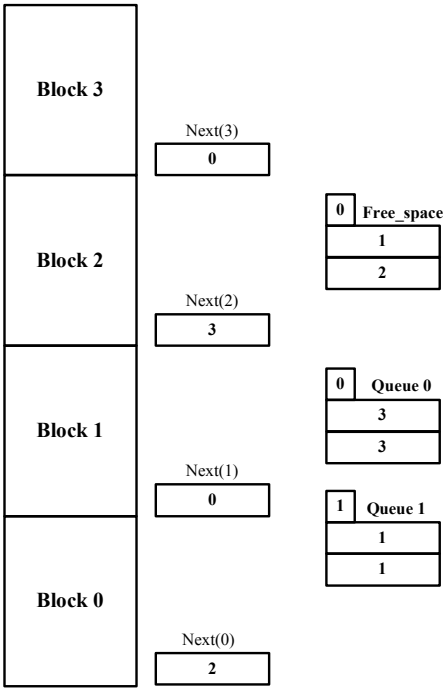
(e)


Block 3
Block 2
Block 1
Block 0
Next(3)
0
Next(2)
3
Next(1)
2
Next(0)
2
0 Free_space
1
1
0 Queue 0
0
3
1 Queue 1
1
1

(f)


Block 3
Block 2
Block 1
Block 0
Next(3)
0
Next(2)
3
Next(1)
0
Next(0)
2
0 Free_space
1
0
0 Queue 0
2
3
1 Queue 1
1
1

(g)


Block 3
Block 2
Block 1
Block 0
Next(3)
0
Next(2)
3
Next(1)
0
Next(0)
2
0 Free_space
1
2
0 Queue 0
3
3
1 Queue 1
1
1

(h)

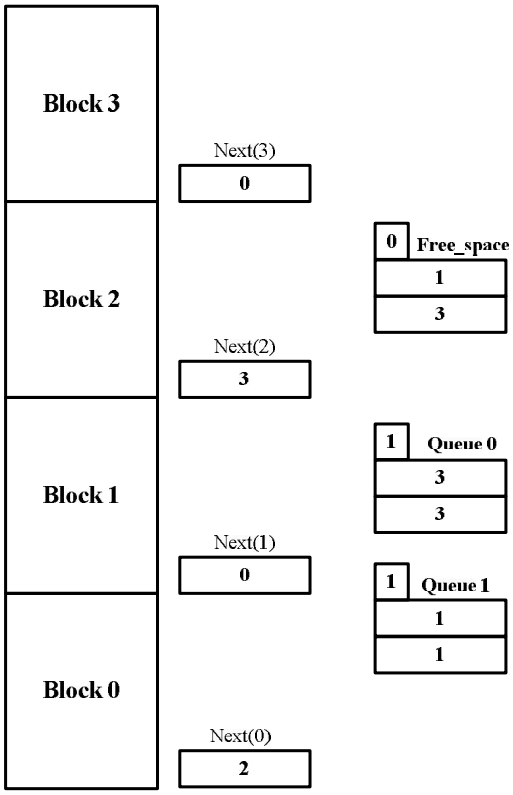


Figure A.1: Example of linked list registers when writing and reading to the buffer.

Figure A.1 (e) shows the state after receiving third packet addressed to queue zero. The packet is put to the head of the free space queue (block three) then moved to be the tail of queue zero. Its tail is now pointing to block three. The next register of the old tail (block two) is now pointing to the new tail (block three). Free space empty flag is set because there is no free block in the buffer.

Figure A.1 (f) shows the state after granting queue one. The packet is read from the head of queue one which was the only packet in the queue. After reading, queue one will be empty (its empty flag will set) and the head block (block one) will be moved to the tail of the free space queue. Because the free space queue was empty, its head and tail will be the same pointing to block one and its empty flag will reset.

Figure A.1 (g) shows the state after granting queue zero. The packet is read from the head of queue zero. After reading, the head of queue zero (block zero) will be moved to the tail of the free space queue. The tail of free space queue will point to block zero and the next register of the old tail (block one) will point to the new tail (block zero).

Figure A.1 (h) shows the state after granting queue zero to the second time. The packet is read from the head of queue zero. After reading, the head of queue zero (block two) will be moved to the tail of the free space queue. The tail of free space queue will point to block two and the next register of the old tail (block zero) will point to the new tail (block two).

Figure A.1 (i) shows the state after granting queue zero to the third time. The packet is read from the head of queue zero which was the only packet in the queue. After reading, queue one will be empty (its empty flag will set) and the head block (block three) will be moved to the tail of the free space queue. The tail of free space queue will point to block three and the next register of the old tail (block two) will point to the new tail (block three). Figure (j) shows the places of head and tail pointers and empty flags.

This Page Intentionally Left Blank

B. Ring Reservation Operation

For $N \times N$ switch, there are one RHE and N CSI's one for each input port. Every CSI checks the destination address of incoming packet with all available addresses of each output port in circulation manner and reserves it if the matched destination address has not been reserved for another input. After the reservation cycle completes, every CSI issues grant signal for the reserved input port in the grant phase. RHE controls the whole process.

An example illustrating a Ring Reservation system is shown in Figure B.1. In this example, since there are four input ports, the arbitration cycle can be completed within four time slots, each time slot means a one period of the main clock of the switch. This scheme uses the serial mechanism. A packet is in each buffer head for four input buffers, the destination address for each packet is 2,3,3,2.

At the initial time slot at Figure B.1 (a), there is only one match between the circulating address (3) in the third CSI and the destination addresses so the circulating token in the third CSI will set.

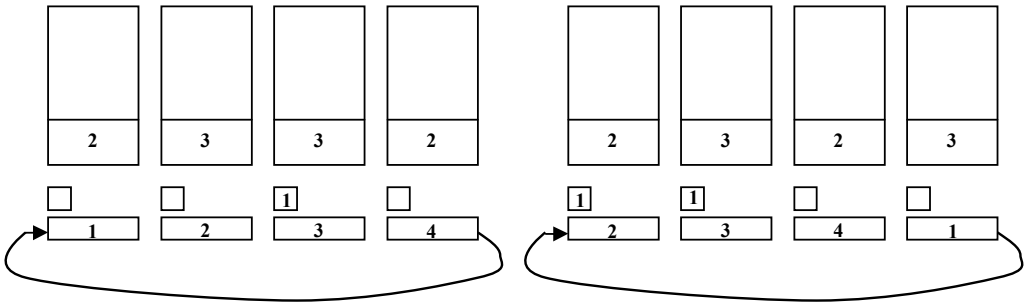
At the first time slot at Figure B.1 (b), all of tokens and circulating addresses will rotate one bit to the left. The circulating addresses in the first and second CSI will match the destination address but circulating address (3) in the second CSI is already reserved where the associated token is set. Hence only the token in the first CSI will set.

At the second time slot at Figure B.1 (c), the circulating addresses and tokens will rotate to the left one bit but there are no matches found.

At the third time slot at Figure B.1 (d), another shift to the circulating addresses and tokens. Both the third and fourth circulating addresses match but the tokens are already set (the output port 2 and 3 are reserved previously).

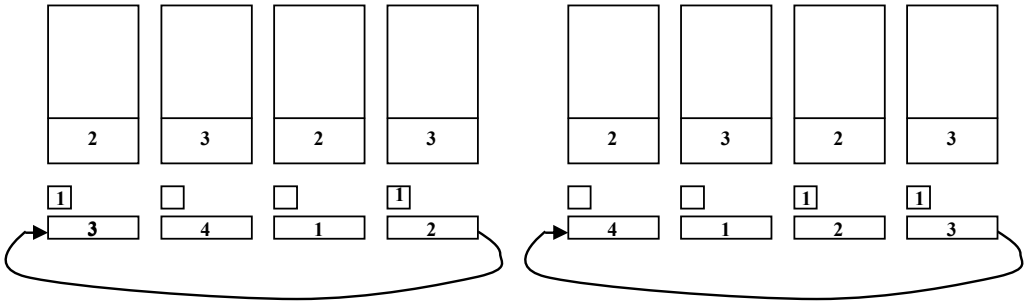
At the fourth time slot at Figure B.1 (e), the circulating addresses and tokens will rotate to the left one bit, every set token will cause the associated CSI to issue a grant to the input buffer to start transmitting the packet to the Batcher Banyan switch. Both input 2 and 3 are granted permission to cross to the Batcher Banyan switch fabric.

At the fifth time slot at Figure B.1 (f), the circulating addresses and tokens will rotate to the left one bit to provide fairness among the input ports where if a buffer contains packets with the same destination address, this cycle will make the ring reservation choose other packets from other buffers not to stick choosing this buffer.



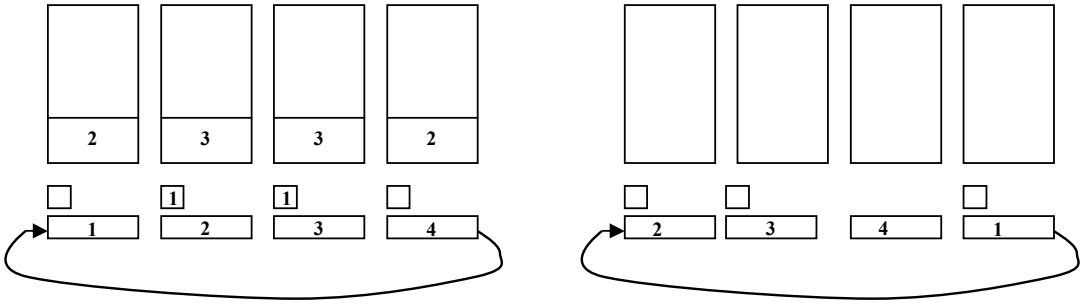
(a): The initial time slot where the circulating address in the third CSI matches, so it's token is set.

(b): The first time slot where the circulating address of the first and second CSI's match, so their tokens are set.



(c): The second time slot. No matches are found.

(d): The third time slot where the circulating address of the third and fourth CSI's match, but their tokens are already set.



(e): The fourth time slot. Issuing grants to the second and third buffers.

(f): The fifth time slot. Additional 0ne circulation for fair reservation. All tokens are reset.

Figure B.1: Example of reservation cycle.

The drawback of the Ring-Reservation is that arbitration cycle has to be done within N clock cycles, where N is the number of input/output ports of the switch. This will represent a bottleneck when the number of ports of the switch is large. This drawback can be neglected if the packet size is larger than or equal to N phits, where phit is the physical bus width or data transmitted in one clock cycle, in other words, the packet has to be transmitted in more than or equal N clock cycles. The main assumption here is that the new arbitration cycle can start quickly after the grants of the previous packets are issued.

This Page Intentionally Left Blank

C. DPA Arbitration Cycle

Figure C.1 shows a practical case example of 4×4 DPA arbitration unit. The two numbers separated by a comma in the arbitration cell represent the input port and the virtual output queue respectively. Bolded squares indicates that the cell is requested ($R(i,j) = 1$). The north and west signals in the first row and column are pulled up (always not asserted) and the east and south output signals in the last column and row are float and not connected to any thing. The arbitration process begins at cell (0,0) which is the highest priority cell and moves diagonally from the top-left to the bottom-right corner of the arbiter. In the second diagonal, both cell (0,1) and (1,0) will be able to issue grants because the higher priority cell (0,0) did not receive a request. In the fourth diagonal, non of cells (1,2), (2,1), or (3,0) is able to issue a grant because of the grants of cells (1,0), (0,1), and (1,0), respectively. Cells (2,2) and (3,3) have no preventing cells to issue a grant. Shaded cells indicate that the cells issued a grant ($G(i,j) = 1$).

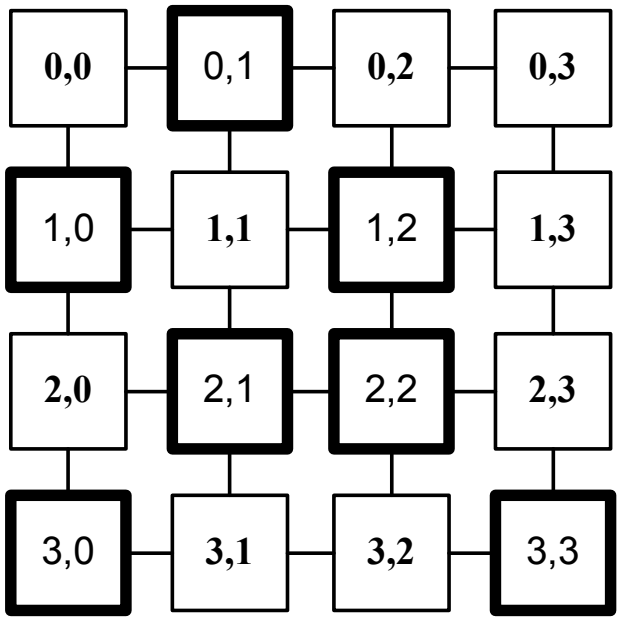


Figure C.1: Example of arbitration cycle of a 4×4 VOQ switch.

Figure C.2 shows the arbitration cycle process step by step. In Figure C.2-(a), the arbitration process starts at the top left cell (0,0). Because it did not receive a request (not a bolded square) it will not issue a grant (light color) and both east and south will not be asserted (thin lines).

Figure C.2-(b) shows the second step which takes place in the second diagonal (cells labeled (0,1) and (1,0)). Both cells receive a request (bolded squares) and their north and west signals are not asserted (thin lines). So, both of them will issue a grant (dark color) and assert their south and east output signals (thick lines).

Figure C.2-(c) shows the third step which takes place in the third diagonal (cells labeled (0,2) , (1,1) and (2,0)). None of them receives a request (not a bolded squares) so none of them will issue a grant (light color). Every south and east will be asserted except the east of cell (2,0) because every north and west is asserted (thick colors) except the west of cell (2,0).

Figure C.2-(d) shows the fourth step takes place in the fourth diagonal (cells labeled (0,3) , (1,2) , (2,1) and (3,0)). All requested cells (cells labeled (1,2), (2,1) and (3,0)) do not issue a grant because they find one or both of the north and west input signals are asserted. Every south and east will be asserted if the north and west is asserted.

Figure C.2-(e) shows the fifth step takes place in the fifth diagonal (cells labeled (1,3) , (2,2) and (3,1)). Cell labeled (2,2) receives a request and both of its north and west input signals are not asserted so it will issue a grant and assert its south an east.

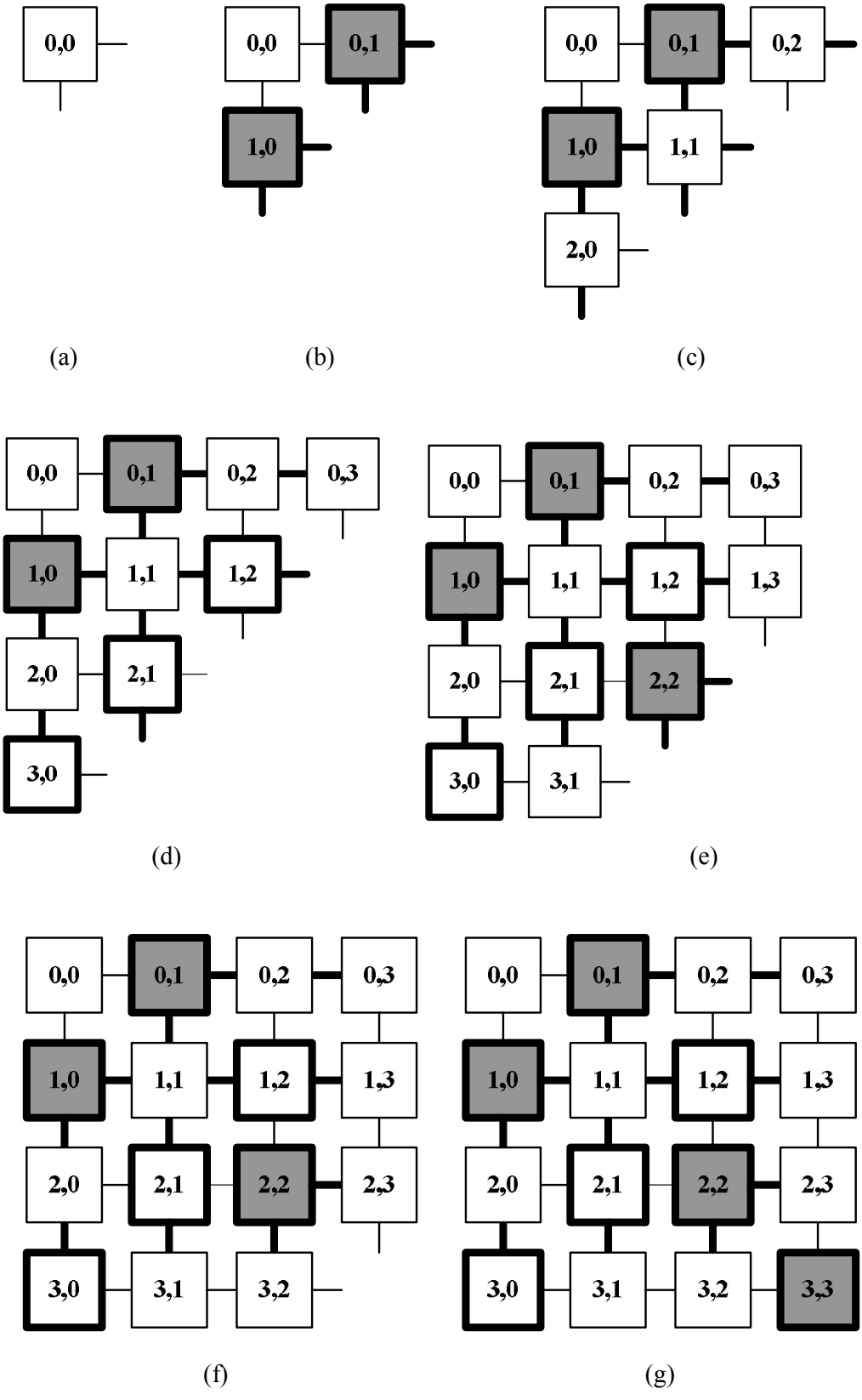


Figure C.2: A step by step example of arbitration cycle.

Figure C.2-(f) shows the sixth step takes place in the sixth diagonal (cells labeled (2,3) and (3,2)). Both are not requested cells so they will not issue a grant. South of cell (2,3) and east of cell (3,2) will not be asserted because their associated north and west respectively are not asserted.

Figure C.2-(g) shows the seventh step takes place in the seventh diagonal (cell labeled (3,3)). The cell is requested and its north and west signals are not asserted. So, it will issue a grant.

تنفيذ وتقييم أداء المحولات الكبيرة
فى شبكات الدوائر المستقبلية متعددة الأنوية

إعداد
أمير حسن محمد زيتون

رسالة مقدمة إلى كلية الهندسة ، جامعة القاهرة
كجزء من متطلبات الحصول على درجة الماجستير
فى هندسة الالكترونيات والاتصالات الكهربية

يعتمد من لجنة الممتحنين:

أ.د/ خالد محمد فؤاد السيد	المشرف الرئيسى
أ.م.د/ حسام على حسن فهمى	المشرف الرئيسى
أ.د/ سراج الدين السيد حبيب	
أ.م.د/ محمد واثق على الخراشى	أستاذ مساعد بقسم هندسة الحاسبات والنظم كلية الهندسة – جامعة عين شمس

كلية الهندسة ، جامعة القاهرة
الجيزة ، جمهورية مصر العربية
2012