1-1-2011

Power and chip-area aware network-on-chip simulation

Masoud Oveis Gharan
Ryerson University

POWER AND CHIP-AREA AWARE NETWORK-ON-CHIP SIMULATION

by

Masoud Oveis Gharan

B.S., Isfahan University of Technology, Iran, 1991

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2011

Author's Declaration

I hereby declare that I am the sole author of this thesis or dissertation.

I authorize Ryerson University to lend this thesis or dissertation to other institutions or individuals for the purpose of scholarly research.

_____

I further authorize Ryerson University to reproduce this thesis or dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

Abstract

Power and Chip-area Aware Network-On-Chip Simulation

Masoud Oveis Gharan, 2011

Master of Applied Science

Electrical and Computer Engineering

Ryerson University

Among different communication architectures employed in Multi-Processor Systems-on-Chip (MPSoC), Network-on-Chip (NoC) is recognized as a state of the art paradigm that can overcome on-chip communication challenges. In this thesis, we introduce the simulation of NoC systems. The structure of a new SystemC based NoC simulator (FANOOS) is presented in this thesis. We discuss various components of the simulator by presenting their SystemC code. We also provide an analytical methodology that employs the micro-architectural level of NoC routers and links by considering their power and chip area requirements. An evaluation flow for early stage design of NoC is introduced. Then different routing mechanisms are analyzed by trading throughput and latency. The ability of our simulator in terms of predicting NoC power and chip-area is demonstrated by investigating the relation between throughput and transactional power consumption for various NoC application benchmarks.

Acknowledgment

After decades far away from academic environment, Ryerson University gave the opportunity to take part on an academic position to upgrade my knowledge and use my experiences. Hereby I truly thank all administrators and staffs who helped and support me in financial and registration matters to do the best my education. I really appreciate the teaching efforts of the professors who taught me during these three academic years, and I really owe my education to them. Here I would like to present my especial thanks to Dr. Gul N. Khan for being an exceptional advisor for my entire graduate career. An advisor and friend, he has managed to perfectly balance encouragement and criticism, and I am in his debt for providing me with the facilities to be a successful researcher in an exciting field. I am astounded by his great breadth of knowledge, and ability to focus on the details and big picture of a project simultaneously. His attention to detail and network-on-chip opinion is an invaluable asset, and I am really thankful to be his student.

Above all, I am thankful for the love and support of my family. I am positively blessed to have a wife, Maryam, and a daughter, Ghazal, as wonderful as mine who have given me an amazing life. Their belief in me was a motivator in everything I did, and their advice and understanding could not be more appreciated.

I owe many thanks to all my friends especially those who are in Micro-system Research Laboratory for their unwavering support and friendship. I love spending time with you guys, some of my best memories have been made here.

Table of Contents

List of Tables

List of Figures

Glossary

2D	two-dimensional
AV	Audio-Video
FAANOS	A flexible and accurate NoC simulator
FCFS	First Come First Serve
FIFO	First-in First-out
FPGA	Field programmable gate array
HP	HOT-POTATO Routing
IP	Intellectual property
LP	LINE_PROBE Routing
MWD	Multi-Window Display
NoC	Network on Chip
OE	ODD_EVEN Routing
PB	Priority Based
PRBB	Priority Based Round Robin
PTMP	point-to-multipoint

RR	Round Robin
SoC	System on Chip
VC	Virtual Channel
VOPD	Video Object Plane Decoder

Chapter 1

Introduction

In VLSI technology, we recently have had a revolutionary development namely System on Chip (SoC) and Network-on-chip (NoC). SoC is the integration of all the components of a computer system on a single integrated circuit (chip). It may contain digital, analog and often radio-frequency functions – all on a single chip substrate [1]. These many interesting systems are too complex to fit on a single chip. There are a lot of challenging overhead, and among them, the two challenges are communication and synchronization between modules [2]. As a solution to these and other problems such as performance, area and power consumption, NoC has emerged. NoC is an emerging paradigm for on chip data transfer of large VLSI systems implemented on a single chip. In an NoC system, modules such as processor cores, memory cores and specialized intellectual property **(**IP**)** blocks exchange data using an on-chip network [1]. NoC replaces dedicated, design-specific interconnection (buses, point-to-point ports, etc.) in SoC with scalable, general purpose network, and it establishes a communication between modules [3].

At every stage of NoC design, there are various options such as topology, switching policy, routing mechanism, traffic pattern, etc. that have impact on the performance, power and area of NoC. Each of the above options has a large design space on its own. Many of these

options are important to the efficiency of over-all SoC systems, and for NoC simulation, all of these options need to be parameterized.

For designers and architects of NoCs, obtaining an optimal NoC is a big challenge due to number of constraints and objectives. One of the best solutions to this challenge is to simulate NoC in a suitable environment. NoC simulators are dedicated testbed frameworks which serve a variety of NoC needs. They simulate relatively fast and inexpensive as compared to the cost and time involved in setting up a test system. NoC simulators allow designers to test NoCs that might be difficult or expensive to emulate using real hardware. For example, simulating the effects of a sudden burst in message flow will be difficult to investigate in hardware setup. Moreover, NoC simulators are particularly useful in allowing designers to test new techniques or amend existing technique. Our work is a modular and extensible SystemC based simulator (FAANOS) [4]. It allows experimenting with various options available at every stage of NoC design such as topology, switching technique, virtual channels and routing methods. FAANOS can be easily extended to include new topologies and routing algorithms. It also produces performance, area and power metrics for a given set of traffic choices.

As the CMOS technology developed at sub-micron levels, the power consumption of an NoC will be a major concern and sometimes may affect the NoC design performance and area. Power dissipation in CMOS circuits is generally classified into two sources: static (or leakage) power dissipation and dynamic (or switching) power dissipation. The estimation of dynamic power dissipation of a chip is a challenging task. In this estimation, one should compute the power when the data transactions happen in the NoC. Two main power models ranging from RTL power estimation tools [5] to early stage architectural power models [6 and 7] have been proposed in the NoC area. In the RTL power estimation, the power models can be obtained from

synthesis and place / route of the RTL code using commercial tools. The generated library of the NoC components is then used during topology synthesis. The shortcomings of this model are the requirement of having complete RTL code and slow simulation (in the order of hours). However, the architectural power model takes only few seconds. This encourages us to choose the architectural power model to incorporate into our NoC simulation framework. This model allows NoC architects to estimate the power in transactional level or maximum condition (when NoC is applied by a full traffic pattern). This property of our NoC simulator is unique as compared with the past similar works.

1.1 Motivation

Two important reasons encourage us to do research on NoC. The first one is the rapid development of NoC that is going to be prevalent in the future of electronic technology. To support our claim, we can say that it is possible to imagine an NoC-based chip as a future field programmable gate array (FPGA). Such a field programmable resource array (FPRA) will use a big number of configurable computation resources by using a programmable NoC platform. Already there have been proposals for designing programmable NoC platforms [8 and 9]. One can easily imagine a scenario where a mesh topology NoC chip, populated with the application-oriented cores, could be available as a mass-produced standard product. A future NoC architecture must be general enough to allow mass production and must have features for configuration to match and meet the applications constrain requirements. Therefore, NoC with such bright future and with many challenging research problems at all levels need further investigation, research and development. For example in terms of power, it is important to understand the different tradeoffs when we balance the energy of on-chip and off-chip

communication for a given energy budget. A better understanding of the area and power tradeoffs will enable the design of more energy and area efficient router architectures, as well as interconnection networks.

The second reason that encourages us is the lack of a homogenous, general and accurate simulator in the NoC area. Most of today's design optimization tools estimate power, performance or both based on architectural aspect of NoC systems. They do not have any facility to estimate the transactional metrics of NoC, or if they have then it is incomplete. This shortcoming leads to the inaccurate or deficient results of power, area and performance metrics. The inefficiency of the simulators that do not create transactional results is obvious. The transactional results give an accurate view of the behaviour of NoC in a real chip that is really important in making early design decisions of NoC. However, for the simulator that have incomplete link to the transactional aspect of NoC, we have noticed that they are not homogenous. In other words, the parts of simulator that estimate performance or power are not homogenously developed (i.e., the performance and power parts are not developed by a group). The users of heterogeneous simulator cannot establish a complete link between different parts of their tool. In this condition, they estimate transactional results from architectural results that is not enough accurate. The other important drawback in most of the today's simulators is their development under software platform that do not have hardware description property (i.e., VHDL and Verilog). By considering the above mentioned points, we developed our simulator in a transaction-level modeling language (SystemC), which deliberately mimics the hardware description languages. This advantage of our simulator enables a designer to synthesize an NoC application like synthesizing a hardware platform, and simulate it using concurrent processes where each process is described using a plain C++ syntax.

1.2 Objectives and Contributions

As we mentioned earlier, the bright future of NoC technology and the lack of a general purpose NoC simulator has encouraged us to investigate NoC simulation. Our work starts with a research on almost all the important layers of NoC simulator. We first develop a homogenous, general and accurate NoC simulation framework. It is homogenous because the backbone of the tool is structured by SystemC language, and all of its parts are developed by our group. It is general because the simulator employs most of the previous approaches and techniques with some new approaches developed during our research. In terms of accuracy of the simulator, we can argue that in such a homogenous and general simulation environment the metric results tend to be accurate. This accomplishment is not achievable without introducing the recent NoC characteristics and issues as well as representing the ability of our simulator by experimental work.

1.3 Thesis Structure

The rest of this thesis is organized as follows. Chapter 2 provides a background on on-chip networks. We introduce the concept, architecture and configuration of NoC as well as propose an analytical methodology for parameterized NoC router components. We conclude this chapter by definition of power, area and performance characteristics and briefly discussing the related work. In chapter 3, we pay attention to the structure of our simulator. The hardware modeling of every NoC module is described in detail, and the working of SystemC process of each module is depicted. This chapter delves into the micro-architecture and circuit-level design of a router synthesized in our simulation framework, and the technological and analytical parameters accompanied by the related formulas for every component are demonstrated. At the

end of this chapter, an evaluation flow is presented and based on that we start an experimental work in chapter 4. We implement two sorts of experiments. In the first experiment set, we configure the simulator and traffic generator to investigate the impact of three different routing algorithms, XY, odd-even (OE), and line-probe (LP) on different NoC applications. In the second experiment set, we first propose a methodology to estimate the maximum length of link in different topologies and based on that we experiment the area and power of some SoC applications. Finally, Chapter 5 concludes this thesis and presents directions for future work.

Chapter 2

Overview

The general structure of NoCs has been presented in the past works [3 and 10]. It consists of cores and switches that are directly connected such that the cores are able to communicate concurrently by sending messages asynchronously. Figure 1 illustrates a typical on-chip network, the on-chip components such as MEMORY, CPU, DSP, RISC, RF interface and I/O that communicate by using an interconnected group of switches.

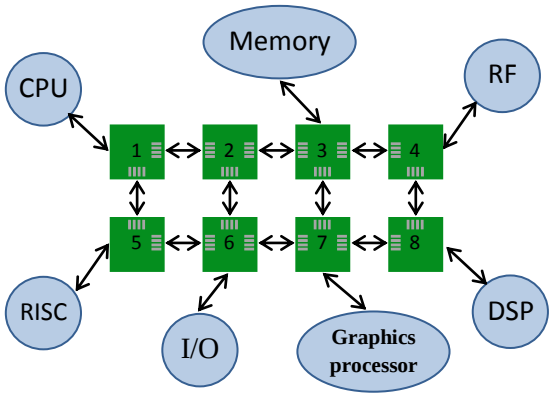


Figure 1: NoC architecture

To illustrate the concept of NoC, we take a detailed look in Fig. 2, where the NoC architecture provides the communication infrastructure for the resources (cores or IPs). It has a bonded structure of elements such as routers or switches, network adapters, and links based on a particular topology [2]. The core can be a sender, a receiver or both and represents SoC modules

such as processor cores, memories or specialized IP blocks. The network adapters couple these cores to the network to create an environment, where the cores send or receive messages asynchronously. In other words, it allows communication from a core to the outside world via a channel. The router consists of different modules such as arbiter, crossbar, demultiplexer, and FIFO or virtual channels. The routers communicate among each other asynchronously to deliver a message to the destination. The link is the communication elements in NoC that connects two routers or a router and a core together. It can be either simple wires or complex communication mechanisms like FIFOs. In the following sections, we introduce the techniques, modules and metrics which are employed in NoC architecture. Section 2.1 describes the architecture of different topologies. The details of messages along with different packet switching methods are discussed in section 2.2. Different routing mechanisms and the architecture of router are presented in sections 2.3 and 2.4 respectively. In sections 2.5 and 2.6 we introduce the details of link, traffic generator, source, and sink modules. Different traffic patterns are discussed in section 2.7. In sections 2.8 and 2.9, we explain the micro-architectural structure of router and link to introduce the model and formulation of the basic electronic elements for estimating power and area. Lastly, we define the power, area and performance metrics in section 2.10 and the related work is discussed in section 2.11.

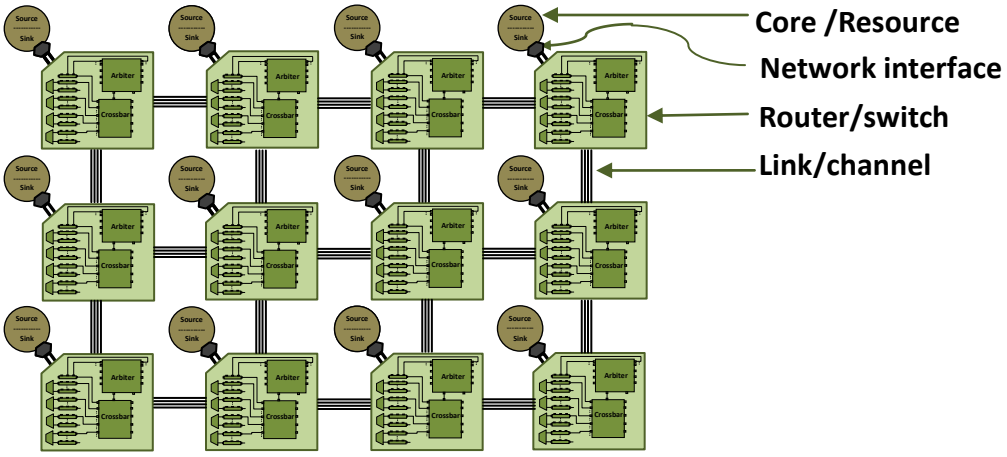


Figure 2: Network

2.1 NoC Topology

An NoC topology is the layout pattern of interconnections of various elements such as links and routers [11]. Generally, NoC topology is classified into two main groups: regular and irregular [12]. The regular topologies are simpler than the irregular ones, and the local interconnections between resources and routers are independent of the network size. Besides, routing in regular topologies is easy that results in potentially small switches, more efficient, high bandwidth, faster clock cycle, and overall scalability [13]. Figure 2.1 depicts different topologies where a, b and c are regular, and d is an irregular topology.

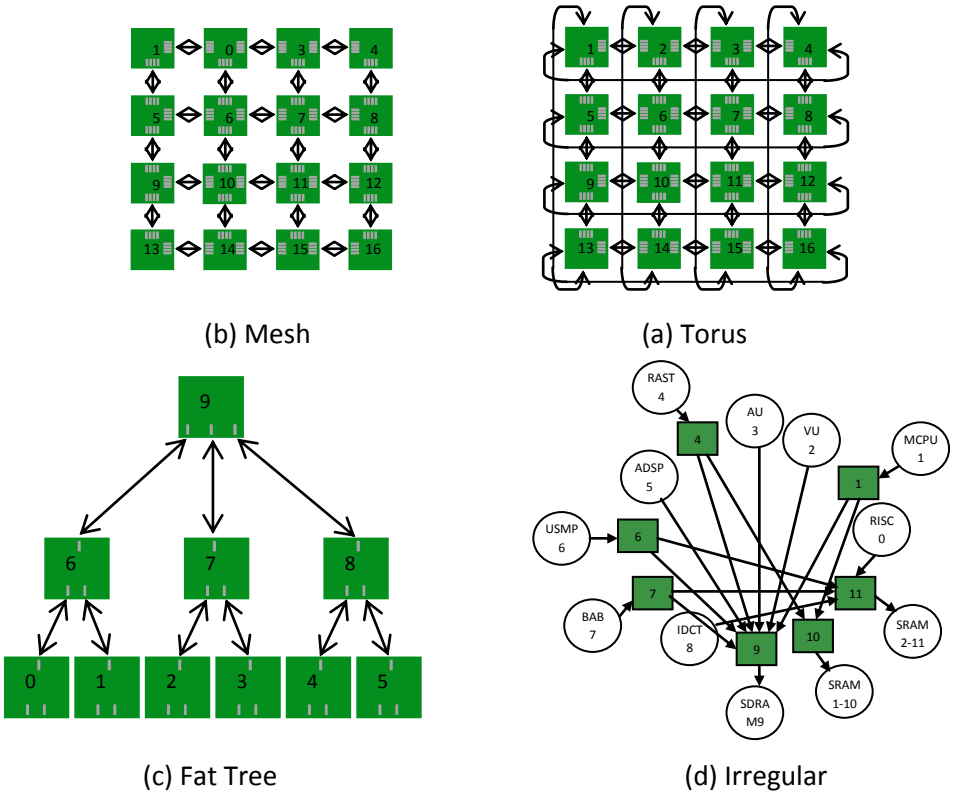


Figure 2.1: Different topologies

2.1.1 2D Mesh/Torus Topology

Mesh and/or torus topology is a regular topology which is most commonly used and studied topology so far [10]. In this topology, each node consists of a core connected to a router.

Each router is connected to four other neighbouring routers by using input and output channels. Each channel consists of two uni-directional point-to-point links between two routers or a core and a router. Routers may have internal queues to handle congestion. Each node is identified by a unique integer called Node_ID. Moreover, each node can be identified by an x and y coordinates. Figure 2.2 depicts 3 × 4 mesh and torus topologies, where each square represents a node in the network.

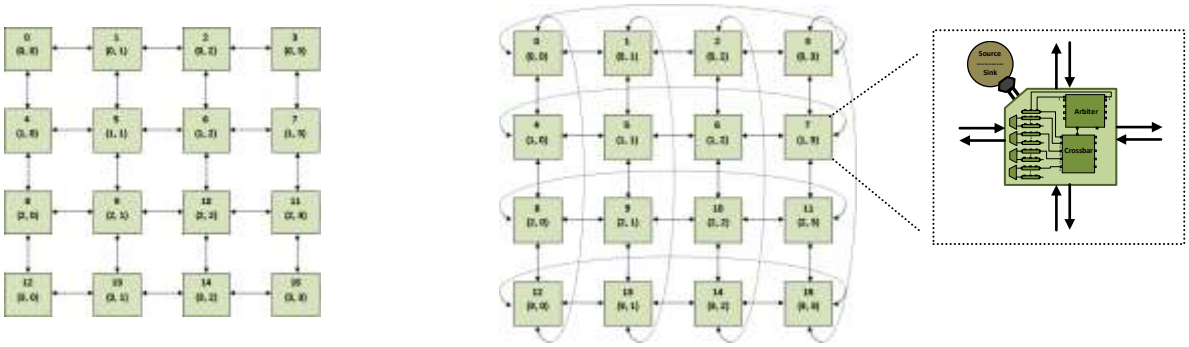


Figure 2.2: Mesh topology (left) and Torus topology (right)

2.1.2 Application Specific Topology

There is a wide variety of irregular topologies in the NoC domain that are also known as application specific topologies. Various researchers consider different layout pattern for it. In this thesis, we introduce application specific topology whose pattern is derived from the specification of an SoC application. Since every application has different specification, the topology derived from that does not have a predefined or well ordered shape, and this leads to limited use of various routing algorithms in application specific topologies. For instance, among various routing mechanisms that are described in section 2.3, only two deterministic routing mechanisms (VIRTUAL CIRCUIT and SOURCE routing) can serve such topologies. To demonstrate the architecture of this topology, we present Fig. 2.3, where an Audio-Video (AV) application is

drawn in the form of nodes and arrows, which is commonly known as core graph [14]. If we put a switch at every node of the core graph, we can have an application specific topology of the application that will be called a core switch graph in this thesis. The router in this topology is modeled to have dedicated output for each input channel. In this way, there is no contention within each router because of the pattern of the topology.

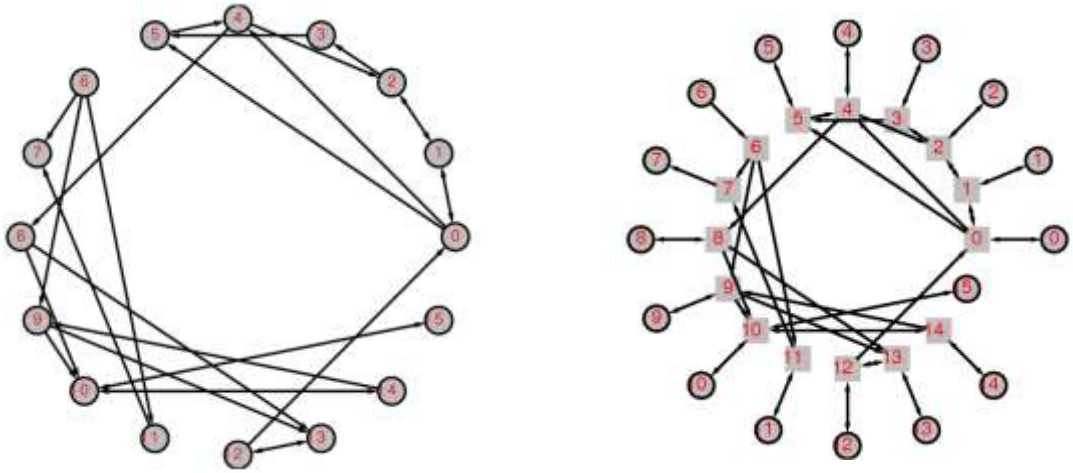


Figure 2.3: Core graph and the core switch graph of AV benchmarking application

2.2 Messages

In the NoC area, a message is an arbitrary amount of information which is sent from a source to a destination. Transferring messages through an NoC can be achieved by two methods: circuit switching and packet switching. In **the circuit switching,** a limited number of dedicated connections are established between routers prior to transferring the message, and there is no need of buffering in this method. **The packet switching** is a newer digital communication method that splits messages into distinct packets that are transmitted independently and then put together into the original message at the destination [15]. The packet in this method is defined as a sequence of binary digits including data and control signals, which is transmitted and switched

as a composite entity [11]. The network in the packet switching can operate a synchronous message passing system. To have a synchronous message passing system between nodes, the nodes need to wait for transfer of message. In other words, the sender will not continue to transfer the message until the receiver absorbs the message. However, there is no need of buffering between the sender and receiver nodes. The message can always be stored at the receiving node, because the sender will not continue until the receiver is ready.

The other advantage of this method is that a channel is occupied during the transmission of the packet only, and upon completion of the transmission, the channel is made available for the transfer of other packets [11]. This leads to an optimized use of the channel capacity, minimize the transmission latency, and increase robustness of communication. Three forms of packet switching such as store-and-forward, virtual cut-through and wormhole are defined in the NoC area.

2.2.1 Store-and-Forward Method

In this method, the whole packet is received and stored at the router, and then forwarded to the next router or destination. In this method, a router needs the buffering capacity of at least one full packet [16].

2.2.2 Virtual cut-Through Method

The router starts forwarding a packet before the whole packet has been received. Normally, as soon as the header information is processed, the packet is transmitted to the destination. This technique reduces latency through the router, but compromises the reliability as compared to store-and-forward method. In this method, a router needs buffering capacity for at

least one full packet. If there is contention in the network, then the packet is stored at the router level [17].

2.2.3 Wormhole Switching Method

In this method, the routing is done as soon as possible similarly to cut-through routing. However, the cut-through flow control assigns buffers and channel bandwidth on a packet level, while wormhole flow control does this at the flit level [1]. The flit (**fl**ow control dig**its**) is defined as a small piece of data which is created from splitting the network packet. It is also defined as the fundamental unit of data transferred in the communication network at the clock rate. The first flit of a packet is called the header flit that holds information about the packet's route (namely the destination address) and sets up the routing path for all subsequent flits of the packet. The header flit is followed by one or more body flits, containing the actual pay load of message as depicted in Fig. 2.4.

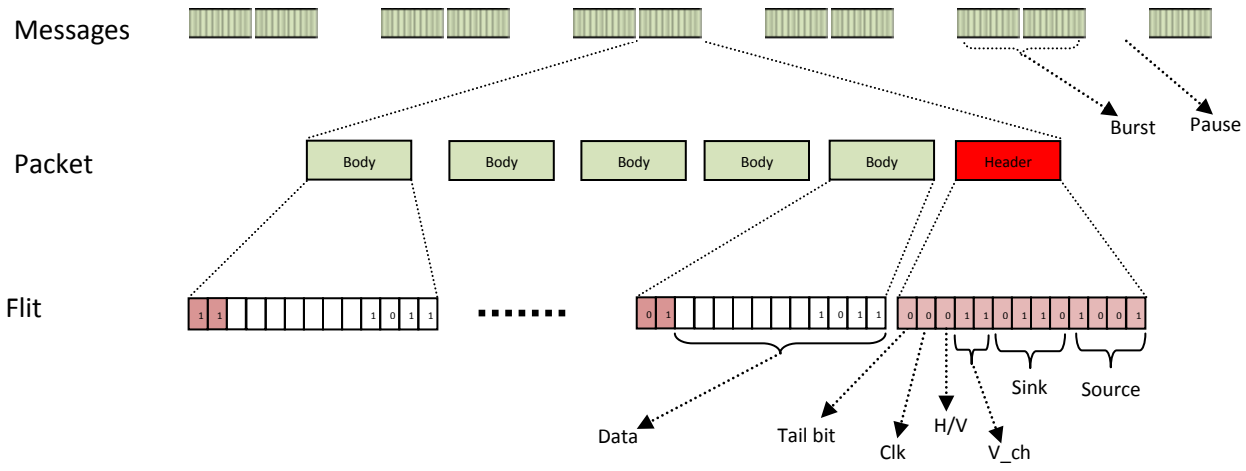


Figure 2.4: Packet message in the NoC

The final body flit called the tail flit performs some tasks to close the connection between two routers. Since a packet is transmitted flit by flit, it may occupy several flit buffers along its path, creating a worm-like image. The wormhole switching is the most popular and well suited for on chip communication, and we have chosen this method for transferring message in our NoC simulation framework. Figure 2.4 provides a detailed description of a message format in this method. The size of messages can have great impact on the performance that is important request in the early design of an NoC [18]. To implement this method, NoC requires a small FIFO flit buffer at each intermediate node. The wormhole flow control also facilitates the implementation of virtual channels. The Virtual-Channel switching is an extension of worm-hole switching, where multiple virtual channels are provided for each input port.

2.2.4 Buffer Implementation

Buffers are a storage medium used to store the message content temporarily due to a difference in rate of flow of data, or time of occurrence of events, when data is transferred from one node to another [11]. The capacity of buffers can affect the network contention as well as cost. The higher buffer capacity leads to the reducing of network contention, thereby improving performance. However, buffers are area and power hungry, and their use should be carefully investigated. The buffers are implemented in a first-in first-out (FIFO) way. In a FIFO, what comes in first is handled first and what comes in next waits until the first is finished, etc. When the packet is small, a suggested optimized size of the FIFO buffer is to be equal to the size of a packet. If the size of the packet is larger than the size of FIFO, then the packet occupies more than a router's buffer, and in the case of contention the situation get worse.

2.2.5 Virtual Channel (VC)

A VC connection is an association established at the asynchronous transfer mode between two or more endpoints for the purpose of resource-switch or switch-switch information transfer [11]. A virtual channel virtually splits a single physical channel into two or more channels, providing two or more virtual paths for the packets to be routed. The VCs improve the performance at the expense of area and power consumption. However, there are several other advantages such as contention alleviation, latency and wire cost reduction. In an NoC environment, the virtual channel is modeled in FIFO buffers and characterized with the parameter "VC".

2.3 Routing Mechanisms

The term routing mechanism refers to the mechanism used to implement any routing algorithm. It is the process of determining and prescribing the path so that a message data can reach the destination, or it is the method to be used for forwarding messages [11]. In an NoC system, routing directs packets from their source toward their destination through the NoC infrastructure. Routing schemes have been classified in several ways in literature. In one of the classifications, they are categorized in deterministic and adaptive groups [18]. The deterministic routing always follows a deterministic path of the network. The examples for such routing mechanisms are SOURCE, VIRTUAL CIRCUIT and XY routing. The adaptive routing mechanisms need more information about the network to avoid congested paths in the network. In adaptive routing implementation, the routing function is described by means of *if-then-else* statements that are synthesized into a combinatorial logic circuit. The examples are OE (odd-even), LP (line-probe) and HOT POTATO. These routing mechanisms are obviously more

complex to implement, and they are more expensive in area, cost and power consumption. The SOURCE, VIRTUAL CIRCUIT, XY, OE, HOT POTATO routing have been introduced and researched in the past works [19, 20 and 12]. The LP routing is a new routing method that is being presented in this thesis. It is being investigated here in the context of NoC simulation.

2.3.1 VIRTUAL CIRCUIT Routing

Virtual circuit routing is a connection oriented communication service that emulates circuit switching. In virtual circuit routing, the packets are delivered by means of packet switching where the connection is established before any packets are transferred and handshaking overhead produce during the connection establishment phase [21]. However, the bit rate and latency service are similar to circuit switching which might be different in a virtual circuit routing due to three main differences. First, the FIFO lengths in the routers vary in the virtual circuit. Secondly, the packet rates generated by the cores are not the same in the virtual circuit. Thirdly, the loads from different cores sharing the same links are different in the virtual circuit. Virtual circuit routing implementation uses a routing table that stores routing information in the form of a set of entries (an entry for each destination), each containing the information about the output port (or the set of output ports) through which the packet must be forwarded to reach its destination. This routing table is constructed and located at the router level in advance when the topology is generated. Therefore, the flit in this routing can be as small as possible and it leads to small channel width and small FIFO buffer.

2.3.2 SOURCE Routing

In SOURSE routing, the sender sends the packet through a partially or completely specified routing path to the destination [12]. SOURCE routing permits comfortable troubleshooting, better trace route, and lets a router to have access to all the possible routes to destination. It also allows a source to have direct effect on performance by directing packets through the network to prevent congestion.

The differences between SOURCE and VIRTUAL CIRCUIT routing is that in SOURCE routing, the routing path (routing table) is constructed by the source, and it is stored in the header flit during run-time. However, in VIRTUAL CIRCUIT routing, the routing path is constructed and located on the router. In this way, the SOURCE routing becomes the most flexible routing solution as it can be updated at run-time (re-configuration) to be adapted to particular scenarios such as the changes in traffic conditions (e.g., multi use-case applications) and the variation of NoC topology (e.g., due to transient or permanent faults) [22]. Although the power dissipation of the routing logic in the router is minimized, the increase in packet size can have negative impact on the performance, resulting in an increase in the energy consumption.

2.3.3 XY Routing

The XY routing is a deterministic routing method for mesh topologies, where the flits are first routed in the X direction until reaching the Y coordinate of the destination, and then it is routed in the Y direction to reach the destination. If a link at the route is in use by another packet, the flit remains blocked in the router until that link is released. The XY routing is simple and easy for mesh routing where a few routers participate in the path routing [20]. In other words, it cannot employ the network capacity fully that leads to increase in the system contention. To

benefit of the features of this routing, we use the XY routing algorithm as the basic routing algorithm for some other routing mechanisms such as OE, LP and HOT POTATO routing in FAANOS.

The XY routing basically works for 2D network topologies like mesh or torus. The mechanism of this routing is simple and easy where the flit must be routed to the destination router address (Dx, Dy) and then to the core port of the router. In other words, in a 2D mesh topology, if each router is identified by its coordinate (x, y) (see Fig. 2.2), the current router address (Cx, Cy) is compared to the destination router address (Dx, Dy) which is stored in the header flit. If the Dx > Cx, flit will be routed to the East port and Dx < Cx, flit will be routed to the West port. For Cx=Dx, the header flit is reached horizontally, and then Dy (vertical) address is compared to the Cy address. If Dy<Cy, flit will be routed to the South port. If Dy>Cy, flit is routed to the North port. For Cy =Dy, the header flit is already in the destination router. When the chosen port is busy, the header flit and all the subsequence flits of the packet should wait in their routers, until the port gets free. Figure 2.5 demonstrates the pseudocode of XY routing algorithm.

```
/*Source router: (Sx, Sy); destination router: (Dx, Dy); current router: (Cx, Cy).*/

begin
        if (Dx>Cx) //eastbound messages
                return E;
        else
                if (Dx<Cx) //westbound messages
                        return W;
                else
                        if (Dx=Cx)
                        { //currently in the same column as destination
                                if (Dy<Cy) //southbound messages
                                        return S;
                                else
                                        if (Dy>Cy) //northbound messages
                                                return N;
                                        else
                                                if (Dy=Cy) //current router is the destination router
                                                        return C;
                        }
end
```

Figure 2.5: Pseudocode of XY routing algorithm

2.3.4 Odd-Even Routing

Odd-Even routing is an adaptive method in which some turning directions are not allowed in order to lower the contention of the NoC system. In a 2D mesh topology, each node is recognized by its coordinate (x, y). In this topology, if the x dimension of nodes of a column is even number, the column is called even, and if the x dimension of nodes of a column is an odd number, the column is called odd. A turn is a 90-degree change of traveling direction which is described as following. There are eight kinds of 90-degree turns that can be associated to each node in a 2D topology. If a turn consists of a change in direction from East to South we call it ES turn. Similarly, the EN, WS, WN, SE, SW, NE and NW turns are defined where E, W, S and N point to East, West, South and North respectively [19]. Generally, the Odd-Even routing is based on two rules. The first rule prevents of EN and ES turns for packets that are in nodes located at the even columns. The second role prevents SW and NW turns for packets that are in nodes

located on the odd columns. Figure 2.6 demonstrates the pseudocode of minimal Odd-Even routing algorithm in which out_dir contains the forwarding directions of the packet.

```
/*Source router: (Sx, Sy); destination router: (Dx, Dy); current router: (Cx, Cy).*/

begin
out_dir < 0;
Ex< Dx-Cx;
Ey< Dy-Cy;
if (Ex=0 && Ey=0) return C;                          //current router is destination
if (Ex=0)
{                                                    //current router in same column as destination
        if (Ey<0)   add S to out_dir;
        else        add N to out_dir;
}
else
{
        if (Ex>0)
        {                                            //eastbound messages
                if (Ey=0)  add E to out_dir;         //current in same row as destination
                else
                {
                        if(Cx % 2 != 0 or Cx=Sx)  //NS turn allowed only in odd column.
                                if(Ey < 0)  add S to out_dir;
                                else        add N to out_dir;
                        if(Dx% 2 != 0 or Ex != 1)  add E to out_dir;
                                                     //allow to go E only if destination is odd column
                                                     // EN and ES turn not allowed in even column
                }
        }
        else
        {                                            // westbound messages
                add W to out_dir;
                if(Cx%2=0)                           //allow to go N/S only in even column because
                                                     // N->W and S->W not allowed in odd column
                        if(Ey<0)          add S to out_dir;
                        else              add N to out_dir;
        }
}
Select a dimension from out_dir to forward the packet.
end
```

Figure 2.6: Pseudocode of OE routing algorithm

2.3.5 Hot Potato Routing

Hot Potato routing (known as deflecting routing) is a decentralized and adaptive routing mechanism, which decreases the need of buffering packets in the routers [23]. The prominent feature of Hot Potato routing is that does not require FIFO buffer in the routers. In wormhole routing, when multiple packets contend for a single outgoing channel, the packets wait at the FIFO buffer level to avoid congestion. However, in Hot Potato routing, when multiple packets want to pass through the same link (the contention link), only one of the packets pass through the link, while the others are routed to other available links, even though the other links do not yield the shortest paths. In a router, every packet has preferred output ports along which it wants to be routed, and when possible a packet is sent along one of these outputs. The implementation of this method creates an environment where the packets are bounced around like a "hot potato" sometimes moving further away from their destination because they have to keep moving through the network. They are constantly transferred until they reach their final destination as the input port cannot keep more than one packet at a time. This technique allows multiple packets to reach their destinations without being dropped. Since it has no FIFO buffer and flow management, a deflection router can be designed with higher speed and lower cost than a wormhole or virtual cut-through switch.

2.3.6 Line Probe Routing

The LP routing is normally used in VLSI technology to design the routing part of an IC (integrated circuit) layout [24]. Figure 2.7 shows the working of this algorithm for a 2D layout of an IC. The line moves from the source in X direction toward the sink. When it reaches to a blocking area it probes the neighbours and chooses one that is in Manhattan distance (the

shortest distance between two points). It turns to Y direction until it reaches to a blocking area. Again it probes the neighbour and chooses one that is Manhattan distance. As you can see it turns around the blocking area until it reaches the sink.

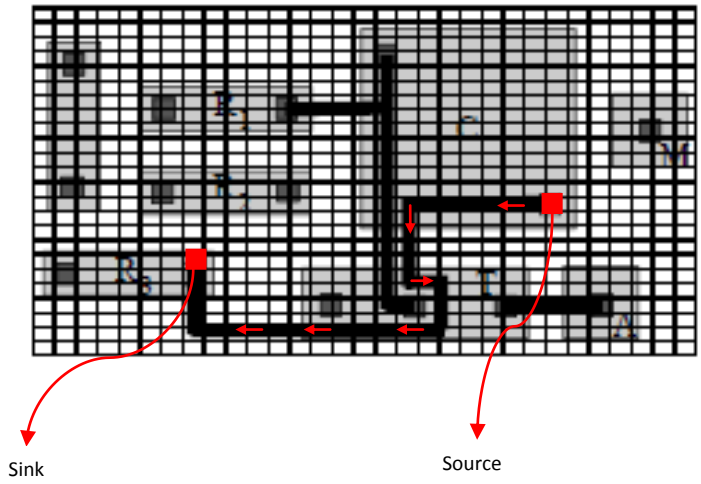


Figure 2.7: an example of layout

Figure 2.8 and 2.9 show a LP-based routing in a 2D mesh topology.

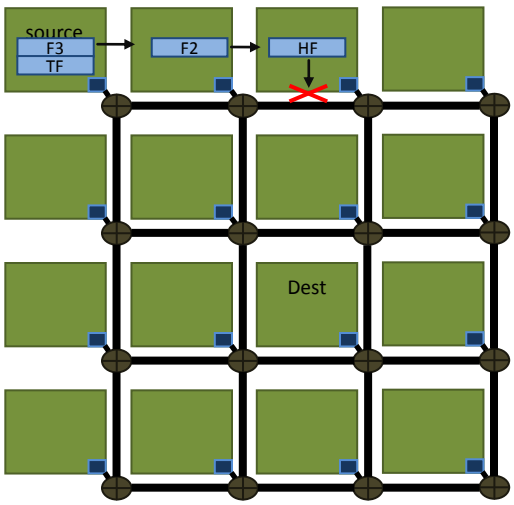


Figure 2.8: a packet is blocked

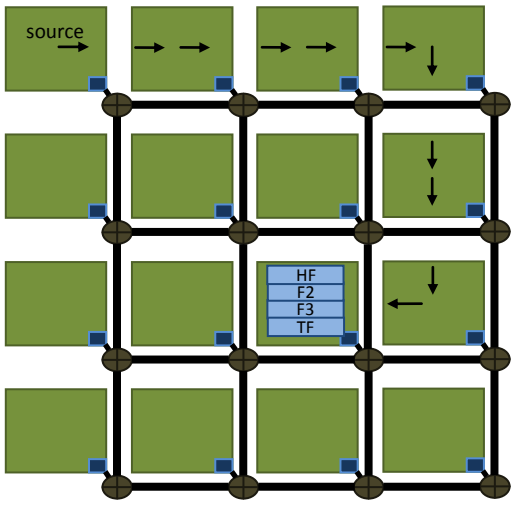


Figure 2.9: Blocked packet is freed by LP algorithm

Compare the LP-routing with the previous routing mechanisms we can say that it has a combined property of wormhole switching and hot potato routing. The switching in this routing

is wormhole as the router can have a FIFO buffer or even virtual channels, but the routing method is like hot potato mechanism. When multiple packets request for a single output, the packets with a lower priority are mis-routed to other output paths. Therefore, due to hot potato property it can choose shorter FIFO buffer without virtual channel. In this way, the router can be designed with higher speed and lower cost, and wormhole switching property prevents or alleviates some problems related to hot potato routing such as high rate of link utilization. In terms of implementation, only one bit extra is used in the header flit, and in the router also LP function can be described with a few *if-then-else* statements. The pseudocode in Fig.2.10 describes the LP algorithm.

```
/*Source router: (Sx, Sy); destination router: (Dx, Dy); current router: (Cx, Cy); current entrance direction
(ent_dir)*/
begin
out_dir= CALL XY algorithm; // call XY algorithm and save the direction in out_dir
temp=out_dir;                   // save this direction to a temp memory
if  out_dir is free             // check the requested output is free
        return out_dir;     // yes, so return this direction
else    //No
{
        out_dir = choose a direction out of out_dir , ent_dir, and local direction;
        if  out_dir is free
                return out_dir;
        else
        {
                out_dir = choose the last direction;
                if  out_dir is free
                        return out_dir;

        }
}
out_dir=temp;     //if all directions are blocked, so stay in first direction for the next clock cycle in router
end
```

Figure 2.10: Pseudocode of LP routing algorithm

2.4 Router

The heart of an NoC is a router or switch. We call it router in this thesis. Routers route and buffer messages between resources (IP core). The design model of router is the same as hardware design, and it is similar to the design presented in Orion NoC design [25]. The router design in Orion is based on the hardware of four real NoC routers: Alpha 21364 router [26], IBM InfiniBand 8-port 12 × switches [27], Intel 80-core Teraflops chip [28], and Intel Scalable Communications Core chip [29]. The other advantage of our router design is its model in terms of power and area parameters that will be introduced later.

The router consists of different modules such as arbiter, virtual channel (VC) allocator, crossbar, demultiplexer, and FIFO or virtual channels. Incoming data (packet) is stored at the input buffers that the VC allocator selects. The arbiter decides to which output port, the input data will be routed. The crossbar gets the data from the input buffers and transfers to the output port which the arbiter assigns. The crossbar is usually implemented as a matrix crossbar.

Two kinds of router models are investigated in this report: regular and irregular. The regular router has five input ports and five output ports as depicted in Fig. 2.11 that is used in 2D topologies such as Torus or Mesh topology. It employs all routing mechanisms that are described in section 2.3. The irregular router has flexible input and output channels and is more effective in irregular topology. Due to its irregular nature, it is limited to utilize every routing mechanism.

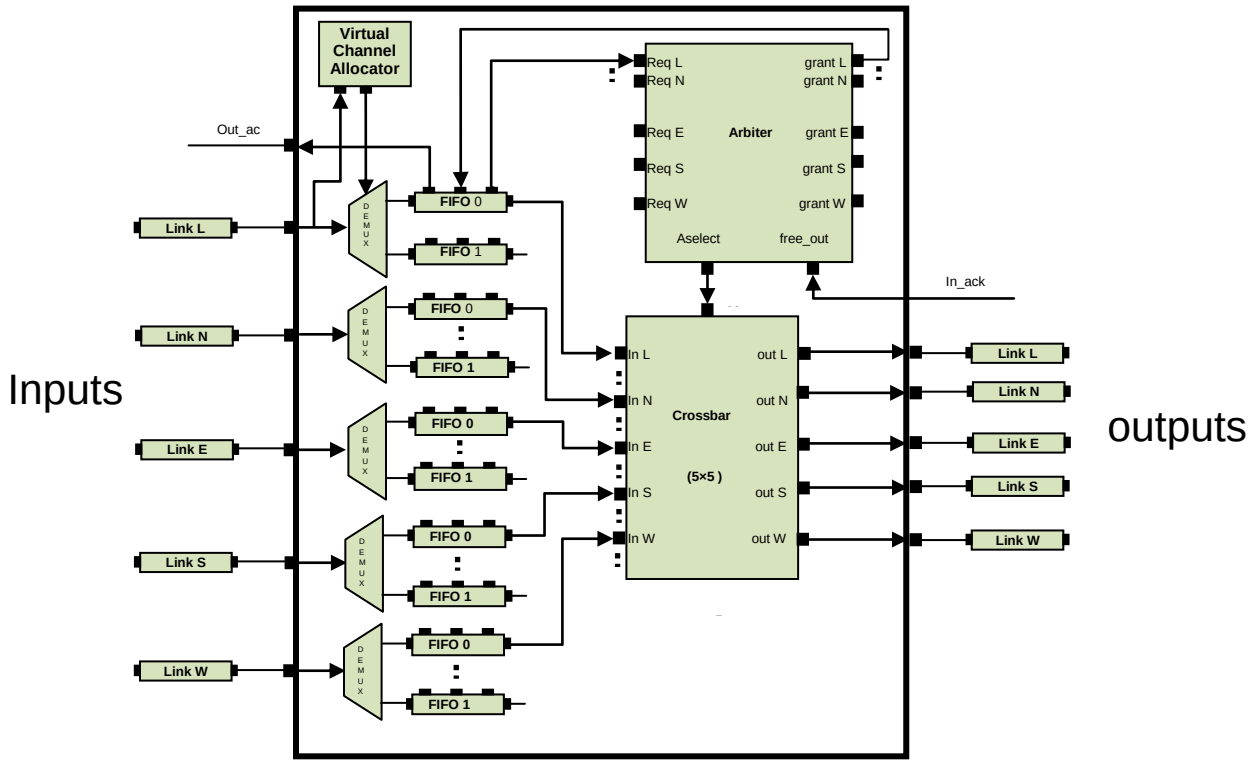


Figure 2.11: 5×5 regular router

2.4.1 Arbitration Techniques

In asynchronous circuits, arbitration selects the order of access to a shared resource among asynchronous requests. Its function is to prevent two operations from occurring concurrently when they should not. For example, in the router that has multiple input messages accessing an output channel, there is a possibility that the requests from two input messages can happen nearly at the same time. The arbiter module must decide that which request to be serviced first. Ivan Sutherland and Jo Ebergen describe arbiters as following [30]. “An arbiter is like a traffic officer at an intersection who decides which car may pass through next. When there are two cars to pass through an intersection at the same time, the traffic officer first stops the cars and decides which car should go first. Then he or she let only one car passes through the

intersection. When the first car passes through completely, the second cars is allowed to pass through. The same situation happens in an arbiter, i.e. when an arbiter gets two requests at once, it must decide which request to grant first. For example, when two packets request access to an output channel at approximately the same time, the arbiter puts the requests into a sequence and grants access to only a packet at a time."

There are various arbitration scheduling mechanisms such as Round Robin (RR), First Come First Serve (FCFS), Priority Based (PB), and Priority Based Round Robin (PRBB) in asynchronous circuits. Usually, RR and FCFS are used for best effort data flits and PB and PBRR are employed for guaranteed traffic.

2.4.2 Round-Robin Scheduling

Round-robin (RR) is one of the simplest scheduling algorithms for processes in a data packet scheduling in NoC, which allocates equal portion of time slices circularly to each process and manage all processes without any priority. RR scheduling is both simple and easy to implement, and it can be starvation-free. However, if the sizes of the jobs of the processes vary considerably, the RR scheduling may not be suitable. In other words, a process that does a big job will consume more time over other processes. In an NoC system, the processes in the arbiter of a router take the same task time, and there is no significant difference between them. Therefore, the round robin will be suitable in an NoC system.

2.5 Link

Links connect routers and cores, and they are implemented as parallel global wires on metal resources. In NoC, the number of links in each channel is set to be equal to the size of a

flit, and a larger flit size can increase link utilization and need more metal resources for power and ground interconnects. Buffered wires are modeled to fit link delay within a single cycle [31].

2.6 Traffic Generator, Source and Sink

We have introduced the architecture of NoC, and now it is time to introduce the modules embedded in our NoC simulation framework. These modules deal with the generation and sending different packet models to assess the effectiveness of an NoC system on three metrics: performance, power and area. In fact these modules imitate the communication behaviour of resources in an SoC system. In terms of sending packet, the resources are classified in two main groups i.e., sender and receiver. The sender is responsible to generate packet based on a given traffic pattern and injects them to the system. In this report, the dummy packets are generated and injected into the system by the source module, and the traffic generator is responsible to dictate the traffic pattern of source module. Therefore, both the source and traffic generator play a key role of sender in our NoC simulation framework. The sink plays the role of a destination/receiver which receives packet and records the number of incoming packets.

2.7 Traffic Pattern

In terms of traffic generation pattern, the traffic can be classified in two patterns: synthetic and application-oriented. In the case of synthetic traffic, the traffic generator generates pattern resembling a simplified packet data model. The sources in the simplified packet data model are generally greedy sources. A greedy source is a traffic model that generates packets at the maximum possible rate and at the earliest opportunity [32]. There will always be packets to transfer, and the source will never be in the idle state due to congestion avoidance or other local

traffic type. When the transfer of a past packet is completed, a new data packet is produced. It means that the queue of the source is never congested.

In terms of the distribution of traffic, synthetic pattern is categorized along three dimensions: spatial distribution, temporal flow distribution and message size distribution. The spatial distribution is associated to the locations of sources and destinations. The temporal flow distribution is related to the message generation probability over time. The message size distribution defines the length of messages to be routed in the NoC.

In application-oriented traffic, the traffic generator generates packet in a pattern that matches with a real application model. It considers all the aspects of traffic such as the rate of packet flow, the burst time or pause time of packet flow, as well as the size of flit and packet. The source and traffic generators should be configured in advance by the users. The following sections discuss further about these patterns.

2.7.1 Spatial Distribution

In spatial distribution, each source core sends packets to different destination cores at each simulation cycle. In other words, the distribution of destinations varies in each simulation cycle. All the sources share the same temporal and size parameters, and the traffic is classified into two categories: the uniform and locality patterns. With the uniform traffic, a network node sends packets to other nodes with the same probability [33]. With the locality traffic, a network node sends packets to its neighbouring nodes with higher probability. In order to build a unified expression for both types of traffic, we define *communication distribution probability* in terms of *communication probability* as follows:

- Communication probability $P_{(i,j)}$ is the probability of sending messages from node *i* to node *j*. Assume the probability *Pc* of sending messages to the network from every node is the same. Therefore, we define Equation 2.1 for any source node *i* with *N* destination.

$$\sum_{j=1}^{N} P_{(i,j)} = Pc \tag{2.1}$$

- Communication distribution probability $Dp_{(i,j)}$ is the probability of distributing messages from node *i* to node *j* where node *i* sends messages to the network. For any source node i with its *N* destination nodes, we define Equation 2.2, meaning that all the messages from node *i* are aimed to all the *N* destination nodes in the network.

$$\sum_{j=1}^{N} DP_{(i,j)} = 1 \tag{2.2}$$

- The distribution coefficient $\alpha_{(d)}$ is the coefficient of distribution of destination around node *i* where *d* is the distance between source *i* and its destination nodes. This array is constant for each NoC system during the simulation and the value of its elements is selected based on the locality of system. Assuming the maximum distance between nodes is *D*, the value of coefficients is defined in Equation 2.3.

$$\sum_{d=0}^{D} \alpha_{(d)} = D \quad 0 < \alpha_i < D \tag{2.3}$$

Now, we define communication distribution probability *Dp* as a relative probability to common probability factor *Pc* in Equation 2.4.

$$Dp = coef \,.\, Pc \tag{2.4}$$

where

- $coef = \frac{\alpha}{D}$

The *coef* is the coefficient probability and when all the elements of α has the same value of one, then the traffic is uniformly distributed due to the uniform distribution of destination for all

the sources. If the value of an element related to a distance has bigger value, the traffic has more tendencies to the nodes related to that distance. Therefore, for a maximum locality distribution (i.e., the sources send message to nearby nodes), the first element of α should have a maximum value of D. To better understand Equation 2.4 an example is given as follows. This example will demonstrate the spatial distribution of destinations in a 2D 4×4 NoC topology. Table 2.1 shows the coefficient distribution (α) and coefficients probability (*coef)* which is based on the distance parameter, *d,* (i.e., the distance between two nodes). If we assume the distance between two nearby nodes is one, then the maximum distance between two nodes will be six. When *d* is equal to zero, then it is the distance between a node and itself. When *d* is equal to one, it is the distance between a node and its first neighbouring nodes and so on. The *coef* is the probability coefficient of sending packets to other nodes. When the *coef* is equal to zero, then the probability of sending packet is also zero. As expected, each node should not send packet to itself. In other words, when d=0 the *coef* is also zero.

Table 2.1 depicts some sample numbers for three traffic patterns of uniform, locality and non-locality. In a uniform traffic, the *coef* for a node to itself is zero and a node to other nodes is constant and equal to 0.16. In full locality traffic, the *coef* for a node to itself is zero, to the first neighbouring nodes is one, and to other nodes is zero. In a non-locality traffic, it is completely in reverse order. The *coef* for a node to itself is zero, to the sixth neighbouring nodes is one, and to other nodes is zero. The other locality conditions will be located between the non-locality and full locality cases.

Table 2.1 Traffic								
TRAFFIC	*d*	0	1	2	3	4	5	6
Uniform	α	0	1	1	1	1	1	1
	Coef	0	0.16	0.16	0.16	0.16	0.16	0.16
Full Locality	α	0	6	0	0	0	0	0
	Coef	0	1	0	0	0	0	0
Non- Locality	α	0	0	0	0	0	0	6
	Coef	0	0	0	0	0	0	1

2.7.2 Temporal Distribution

From the temporal distribution point of view, traffic can be classified into two categories of bursty pattern and continues pattern. In a continuous traffic pattern, the source module continuously sends packets on a regular interval as close as possible to the bit rate specified, where the source is a greedy source. The bursty traffic pattern sends a burst of data at the maximum specified burst rate, and then pauses for a specific amount of time. In other words, it can be represented by an alternating on and off time periods. During the on period, packet is generated in fixed intervals, and during off, no packet is generated.

2.7.3 Message size distribution

The message size specification defines the length of communicated messages. There are two variables for changing the size of message, the size of flit and packet. The size of a flit also determines the width of buffers in the router. Therefore, a small flit needs the small buffer size leading to a small size of the router and consequently lower amount of resources and links. The size of packet that is made of flits does not have effect on the structure of NoC.

2.7.4 Source-by-Source Distribution

Source-by-source traffic is a combinational traffic pattern of two distribution patterns: the temporal flow and message size. In this traffic pattern, each source is configured depending on its own temporal and size parameters. In addition, it statically defines communication before the simulation starts, implying that the destination nodes are fixed for source node during the entire network simulation. The temporal characteristics and message size specification of each node can be approximated using communication traces. This type of traffic is used to construct application-oriented workloads.

2.7.5 Poisson Traffic Distribution

Another simplified traditional traffic generation model in the network technology is the Poisson process where the number of incoming messages follows a Poisson distribution.

2.7.6 Application-Oriented Pattern

This pattern is a subset of source-by-source distribution, where the packet flow mimics the communication characteristics of a given application. Therefore, each source is configured in accordance with its constraints at the application. The destination nodes are fixed for the source node during the entire network simulation. The temporal characteristics and the message size specification of each node are defined and they can be varied depending on the limitation in the application. Every source node should be defined in advance in terms of its destination, temporal characteristics and message size specification. This information can be derived and provided in form of a set of text entries from the application.

2.8 Power modeling

As the CMOS technology developed under sub-micron levels, the power consumption of an NoC design has become a major concern and sometimes dominates the design performance and area. Power dissipation in CMOS circuits can be classified in two sources: static (or leakage) power dissipation and dynamic (or switching) power dissipation. Static power is mainly dependent on the required chip resources, whereas dynamic power is determined by the switching activity. Dynamic power is still the major component of the overall power consumption during active operations. An enormous power dissipation of the CMOS circuits is consumed only when the transistors are switching. In context of NoC design, we need to take care of static power, since it has already been consuming a considerable portion of the active power in recent process technologies. It will further increase while dynamic power becomes smaller when the technology is scaled down. Different types of energy models have been used for the calculation of energy requirements of NoC systems. These models are commonly based on static and dynamic power estimates from synthesized gate-level models of network components. The following sections introduce an analytical method which consider the micro architecture of basic components of an NoC system to estimate the overall power consumption.

2.8.1 Static Power Modeling Methodology

Static power dissipation is currently one of the main factors limiting the performance of a computer system. It is formulated as $P = I_{static} . V_{dd}$ where I_{static} is the static current and V_{dd} is the supply voltage. Therefore, the static power dissipation in each clock cycle is formulated as:

$P = I_{static} V_{dd}/F_{clk}$ where F_{clk} is the clock frequency. We can calculate I_{static} for each component of the NoC system i.e., based on architectural and technological parameters of each component of

the system. However, V_{dd} and F_{clk} are the input parameters which should be specified by users in network simulation. I_{static} or leakage current has five basic sources [34]:

- sub-threshold leakage current
- Gate tunnelling current
- Reverse biased pn junction current
- Punch through current
- Gated induced drain leakage.

The sub-threshold condition happens when a transistor is OFF. It is due to the gate–source threshold voltage of the transistors is below the supply voltage of CMOS design (V_{dd} might have been 5 V and V_{th} for both NMOS and PMOS might have been 700 mV). Tunnelling Current through SiO_2 (Gate Oxide)is very small but at very small thickness levels, electrons can tunnel across the very thin insulation. Tunnelling current becomes very important for transistors below 130nm technology. There is a small reverse leakage current, which is formed due to formation of reverse biased between pn junctions. It is very small as compared to sub threshold and tunnelling currents, however, it may be ignored during power calculations. Two last sources are also very small as compared to sub threshold and tunnelling currents, and they will be ignored.

For estimating leakage power, an architectural approach is proposed by Chen and Peh [34] where the leakage current resources have an almost linear relation with the transistor width. For instance, sub threshold current I_{sub}, which currently dominates leakage current is defined as follows:

$$I_{sub}=I_o\left[1-e^{\left(-\frac{V_{ds}}{V_t}\right)}\right]e^{\left(\frac{V_{gs}-V_{th}-\acute{V}_{off}}{nV_t}\right)} \quad (2.5)$$

$$I_o=\mu\frac{W}{L}\sqrt{\frac{q\varepsilon_{si}.NDEP}{2\varphi_s}V_t^2} \quad (2.6)$$

To derive an architectural leakage power model, we can separate the technology-independent variables such as transistor width from those that stay invariant for a specific process technology:

$$I_{leak}(i, s) = \frac{W(type(i,s))}{L} \cdot I'_{leak}(i,s) \quad (2.7)$$

where

- I_{leak} is total leakage current;
- I'_{leak} is leakage current per unit transistor width over length;
- $W(type(i; s))$ refers to the transistor width of NMOS when NMOS determines the leakage current (*i.e.* $type(i; s)$ is N), or PMOS (when $type(i; s)$ is P).

Orion 2.0 [25] follows the same approach as of Chen and Peh [34], however, it adjusts the approach as follows. Chen and Peh only considered sub-threshold leakage, whereas from 65nm and beyond, gate leakage gains importance and becomes a significant portion of the leakage power. This is even more visible for high-performance applications where gate oxides are much thinner (i.e., 1.5nm in 65nm high-performance library). We will follow the same methodology proposed in Orion 2.0, and our architectural leakage power model is formulated by Equation 2.8.

$$I_{leak}(i, s) = W(i, s) \cdot (I_{sub}(i, s) + I_{gate}(i, s)) \quad (2.8)$$

where

- I_{leak} is total leakage current;
- I_{sub} and I_{gate} are sub-threshold and gate leakage currents per unit transistor width for a specific technology, respectively;
- W(i, s) refers to the effective transistor width of component *i* at state *s*.

I_{sub} and I_{gate} are measured by using HSPICE simulation. Then we compose architectural leakage power model in a bottom-up fashion for each building block [34].

2.8.2 Dynamic Power Dissipation

The estimation of dynamic power dissipation of a chip is a challenging task in the design of chip. For the estimation of NoC power we should compute the power when the chip is doing transactions. The dynamic power dissipation has two basic elements: switching dissipation and short circuit dissipation. In switching power dissipation, the CMOS circuit dissipates power by charging the various load capacitances (mostly gate and wire capacitance as well as drain and some source capacitances) whenever they are switched. In one complete cycle of CMOS logic, current flows from V_{DD} to the load capacitance to charge it and then flows from the charged load capacitance to GND during discharge. In short circuit power dissipation, as there is a finite rise/fall time for both pMOS and nMOS during transition (say from *OFF* to *ON*), the transistors will be *ON* for a small period of time in which current will find a path directly from V_{DD} to GND i.e. short circuit. Short circuit dissipation can be minimized by matching the rise/fall times of the input and output signals and it is a small function of switching dissipation. This element of dynamic power is ignorable in estimation of power. In this thesis, dynamic power is estimated based on switching power dissipation and it is formulated by:

$$P = \frac{1}{2}\alpha.C.V^2{}_{dd}.f_{clk} \quad (2.9)$$

where

- α is the switching activity;
- C is the switch capacitance;
- V_{dd} and f_{clk} are the supply voltage and clock frequency respectively.

The detailed parameterized equations for estimating switch capacitance C of each component of an interconnection network are derived from different resources like Cacti [35] and the switching activity α of these components are tracked through network simulation.

2.8.3 Analytical Methodology

Our proposed analytical method is derived by decomposing the circuit into many equivalent RC circuits, and using simple RC equations to estimate the area and power of each stage [35]. In the following sections, we decompose the structure of smallest component of a CMOS circuit, a transistor and describe how resistances and capacitances are estimated as well as how they are combined and the power of a stage calculated.

A Typical Transistor Layout

Figure 2.12 and 2.13 show typical transistor layouts for small and large transistors respectively and Fig. 2.14 depicts two stacked transistors. It has assumed that if the transistor width is larger than 10 µm, the transistor is split as shown in Fig. 2.13.

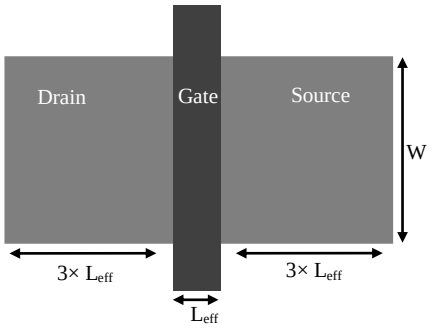


Figure 2.12: Transistor geometry if width < 10 µm

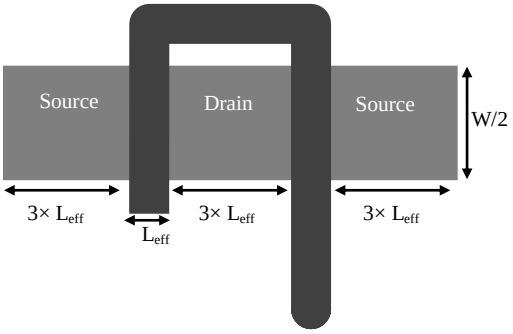


Figure 2.13: Transistor geometry if width >=10 µm

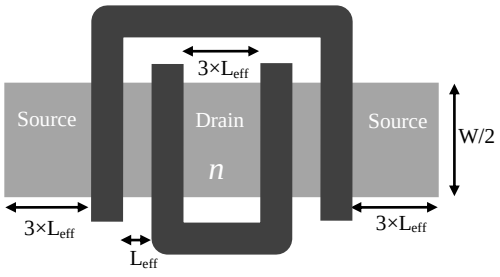


Figure 2.14: Two stacked transistor if width >=10 µm

2.8.4 Gate Capacitances

The gate capacitance of a transistor consists of two parts: the capacitance of the gate, and the capacitance of the poly silicon line going into the gate. By considering Fig. 2.12, if L_{eff} is the effective length of a transistor, *Lpoly* is the length of the poly line going into the gate, C_{gate} is the capacitance of the gate per unit area, and *Cpolywire* is the poly line capacitance per unit area, then a transistor of width *W* has a gate capacitance of:

$$gatecap = W.\ L_{eff}\ .C_{gate} + L_{poly}\ .\ L_{eff}\ .\ C_{polywire} \tag{2.9}$$

The same formula holds for both NMOS and PMOS transistors. The value of *Cgate* depends on whether the transistor is being used as a pass transistor (i.e. the device may conduct current in either direction) or as a pull-up or pull-down transistor in a static gate. Thus, two equations for the gate capacitance are given below.

$$gatecap\ (W,L_{poly}) = W.\ L_{eff}\ .C_{gate} + L_{poly}\ .L_{eff}\ .C_{polywire} \tag{2.10}$$

$$gatecappass\ (W,L_{poly}) = W.\ L_{eff}\ .C_{gate,pass} + L_{poly}\ .\ L_{eff}\ .C_{polywire} \tag{2.11}$$

The values for C_{gate}, $C_{gate,pass}$, $C_{polywire}$, and L_{eff} are derived from Cacti [35]. A different L_{poly} was used in the model for each transistor. L_{poly} was chosen based on typical poly wire lengths for the structure in which it is used.

2.8.5 Drain Capacitances

The drain capacitance is composed of both an area and perimeter component. Using the geometries in Fig. 2.12 and 2.13, the drain capacitance for a single transistor can be obtained. If the width is less than 10 µm,

$$draincap(W) = 3\ L_{eff}\ .\ W\ .\ C_{diffarea} + (6\ L_{eff} + W)\ .\ C_{diffside} + W.C_{diffgate} \tag{2.12}$$

Where

- $C_{diffarea}$, $C_{diffside}$, and $C_{diffgate}$ are process dependent parameters (there are two values for each of these: one for NMOS and one for PMOS transistors).

$C_{diffgate}$ includes the junction capacitance at the gate/diffusion edge as well as the oxide capacitance due to the gate/source or gate/drain overlap. Values for n-channel and p-channel $C_{diffgate}$ are also derived from Cacti [35]. If the width is larger than 10 µm, it is assumed that the transistor is folded (see Fig. 2.13) reducing the drain capacitance to:

$$draincap(W) = 3\, L_{eff} \,.\, W/\, 2\, .C_{diffarea} + 6\, L_{eff}\, .C_{diffside} + W\, .\, C_{diffgate} \quad (2.13)$$

Consider two transistors (with widths less than 10 µm) connected in series, with only a single *Leff.W* wide region acting as both the source of the first transistor and the drain of the second. If the first transistor is on and the second transistor is off, the capacitance is seen by looking into the drain of the first one is:

$$draincap(W) = 4\, L_{eff} \,.\, W\, .\, C_{diffarea} + (\, 8\, L_{eff} + W).\, C_{diffside} + 3W\, .\, C_{diffgate} \quad (2.14)$$

Figure 2.14 shows the situation if the transistors are wider than 10 µm. In this case, the capacitance seen looking into the drain of the inner transistor (*n* in the figure) assuming it is on, but the outer transistor is off is:

$$draincap(W) = 5\, L_{eff}\, .W/\, 2\, .\, C_{diffarea} + 10\, L_{eff}\, .\, C_{diffside} + 3W.\, C_{diffgate} \quad (2.15)$$

To summarize, the drain capacitance of *n* stacked transistors is:

if $W >= 10$ µm

$$draincap(W,n) = 3\, L_{eff}\, .W/\, 2\, .C_{diffarea} + 6\, L_{eff}\, .C_{diffside} + W.C_{diffgate} +$$

$$(\, n - 1\,) \,.\, \{\, L_{eff}\, .W.C_{diffarea} + 4\, L_{eff}\, .C_{diffside} + 2W.C_{diffgate}\} \quad (2.16)$$

if $W<10\mu m$

$$draincap(W,n) = 3\, L_{eff}\, .W.C_{diffarea} + (\, 6\, L_{eff} + W)\, .C_{diffside} + W.C_{diffgate} +$$

$$(\, n - 1\,) \,.\, \{\, L_{eff}\, .W.C_{diffarea} + 2\, L_{eff}\, .C_{diffside} + 2W.C_{diffgate}\} \quad (2.17)$$

2.9 Area Modeling

With the increasing cores on a single NoC design, the area occupied by the communication components such as links and routers will also increase. As area is an important economic incentive in IC (integrated circuit) design, it needs to be estimated early in the design flow to enable design space exploration. We employ a similar model and analysis as used in Orion 2.0 to estimate the areas of transistors and logic gates. Orion 2.0 also used a model described by Yoshida et al. [36] and the analysis put forward by Thoziyoor et al [35]. This is a fast technique to estimate standard cell characteristics before the cells are laid out. Figure 2.15 shows the layout model that has been used by Yoshida et al [36]. Table 2.2 shows the process and technology-level input parameters required by this gate area model. When a transistor exceeds a certain maximum value, the transistor assumed to be folded. Given the width of an NMOS transistor as *Wn*, the number of folded transistors can be calculated as follows:

$$N_{folder\_transistor} = \frac{W_n}{H_{n\_diff}} \quad (2.18)$$

The total diffusion width of $N_{stacked}$ transistors when they are not folded is given by Equation (2.19).

$$W_{diffusion\text{-}area} = 2(W_{contact}+2S_{pc})+N_{stacked}\,W_{poly} + (N_{stacked}-1)S_{pp} \quad (2.19)$$

The Total diffusion width of $N_{stacked}$ transistors when they are folded is given by Equation (2.20).

$$W_{diffusion\text{-}area} = N_{folder\_transistor}(\,2(W_{contact}+2S_{pc})+N_{stacked}\,W_{poly} + (N_{stacked}-1)S_{pp}\,) \quad (2.20)$$

Finally the height of a gate is calculated using Equation (2.21).

$$H_{gate} = H_{n\text{-}diff} + H_{p\text{-}diff} + H_{gap\text{-}opp} + 2H_{power\text{-}rail} \quad (2.21)$$

Table 2.2 Process and technology input parameters used in the gate area model [35]	
Parameter	Description
H_{n_diff}	Maximum height of *n*-diffusion
H_{p_diff}	Maximum height of *p*-diffusion
H_{gap_same}	Minimum gap between diffusions of the same type
H_{gap_opp}	Minimum gap between *n* and *p* diffusions
H_{power_rail}	Height of *Vdd* and *Vss* rails
W_{poly}	Minimum width of poly
S_{pp} Minimum	poly-to-poly spacing
$W_{contact}$	Contact width
S_{pc} Minimum	poly-to-contact spacing

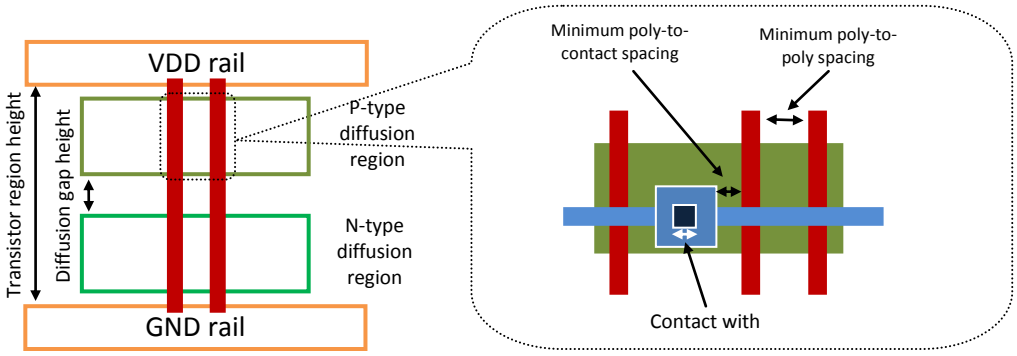


Figure 2.15: Layout model of gates [38]

2.10 Power, Area and Performance Metrics

NoC designers and researchers are often interested in knowing the expected performance, power and area of their system. From the user point of view, this is often said as either "which system will implement my data that is most effective for my needs?" or "which system will convey the most data per unit cost?" NoC designers are interested in choosing the most effective architecture or design constraints which implements its final constrains. In most cases, the benchmark of what an NoC system can do, or its "maximum performance" is what a user designer will be interested. In this section, we intent to describe the definition of some important metrics related to performance, power and area that are used in the NoC area and especially in

our NoC simulation work to be a supportive of the trade-offs, while will be presented in this thesis.

In terms of performance, there are four characteristics such as throughput, latency, packet drop and link utilization that have important role for choosing the most effective design. In terms of power, there are four characteristics such as static, dynamic, architectural and transactional power estimation that easily describe the power behaviour of NoC design. In terms of area, we do not propose extra metric. The followings are a short description of these metrics.

2.10.1 Throughput

The maximum data throughput rate of a system is the first important performance characteristic that designers are often keen to measure. When examining throughput, the term 'Maximum Throughput' is frequently used. In NoC design, a typical method to measure the throughput is to deliver successfully a large number of packets and measure the time taken for delivery. The throughput is calculated by dividing the number of packets by the time in order to find the throughput in terms of packet per second. The throughput is also the average rate of successful packets transferred over a communication channel. This data may be transferred over a physical or logical link, or pass through a certain network router. The throughput is usually measured in data packets per second or data packets per time slot (the time slot is equal to simulation cycles in our work). The system throughput or the aggregate throughput is the sum of the data rates that are transferred to all routers in an NoC. In this thesis, we measure the throughput in percentage. In other words, the average rate of successful packets transferred divided by the maximum rate of packet flow in the network.

2.10.2 Latency

The second important performance metric is latency. Latency refers to a short period of delay between the packet entry time and when its emergence from the system. A fundamental factor in latency is known as delay. Network delay is also an important design and performance characteristic of the NoC. The delay of a network specifies how long it takes for a packet to be transmitted across the network from one point to another. It is typically measured in multiples or fractions of seconds. Delay may differ slightly, depending on the location of the specific pair of communicating cores. Although users only care about the total delay of a network, for the sake of better understanding of function of our NoCsimulator, we divide the delay into several parts [37]:

- Processing delay – time which the routers take to process the packet header
- Queuing delay – time which the packet spends in FIFO
- Transmission delay - time which takes to push the packet's flits onto the link
- Propagation delay - time for a packet to reach its destination

In our NoC simulator, the processing delay is ignored as insignificant in comparison with other forms of network delay. Due to which, our simulator does not perform complex algorithms and examining or modifying the packet content. The queuing delay is the time a packet waits in a FIFO until it can be transmitted. It is a key component of NoC delay. When packets arrive at a router, they have to be processed and transmitted. A router can only process one packet for each output channel at a time. If packets arrive faster than the router can process them, the router puts them into the FIFO until it gets time to transmit them. Transmission delay is the amount of time required to push all of the packet's flits into the link. In other words, transmission delay is a function of the packet's length and has no relation with the NoC architecture. Therefore, we

ignore it in our simulator. Propagation delay is the medium amount of time taking for the header flit to travel from the source core to the destination. The combination of propagation, transmission, queuing, and processing delays produces the latency of an NoC.

In FAANOS, the measurement of delay is a little simpler. The routers are stimulated by flit event, and when a flit reaches the router, it starts delivering to local sink or nearby routers. In our simulator this deliverance does not have latency. Therefore, the latency a network creates for a packet is zero. If a flit reaches a router and the router cannot deliver it at that event, the latency will be created. Latency depends on how many clock cycles the flit is waited on the router to be delivered.

2.10.3 Packet Drop

The third performance metric is the packet drop. Dropping packets in computer networks is different with NoC technology. In computer networks, packets may be dropped when the network devices are overwhelmed. However, in NoC design, we cannot allow packet dropping in the router. We use this term to have a measure of packet generation in each source module. In a simplified packet data model, each source always has data to transmit and is never in an idle state due to congestion. A new data-packet is generated when the transmission of previous packet is completed, meaning that the sender is never congested. Therefore, in congestion condition, the source might drop some packets. This characteristic of NoC design helps designers to have a measure of the rate of generation of packets in the sources on different location of the topology. For this reason, each source has an individual record of dropped packets in output results

2.10.4 Link Utilization

The fourth performance metric is the link utilization. It is the proportion of the NoC links which is used by the traffic that arrives at it. It should be strictly less than one for the system to function well. Lower link utilization leads to the more effective systems. This characteristic affects indirectly on two other important metrics such as delay and power consumption. On measuring a delay, we did not consider the delay related to passing the packet through the link. If it is considered, in this matter, then higher utilization link leads to more delay. Moreover, in terms of power metric, higher utilization link means more movement of packets in the network that leads to higher consumption of power in the system. The link utilization also can be used to calculate delay and power consumption to compensate their errors. For example, if the frequency of a system is high, then the delay time of links are considerable, and constant proportion of the link utilization can be added to the total delay to compensate. This methodology can also be used to compensate the error of power consumption.

2.10.5 Static and Dynamic Power Consumption

As mentioned earlier, static power consumption is a function of leakage current and leakage current is stagnant during transaction level of system. However, dynamic power consumption is different. In order to highlight this difference we define two metrics for power estimation namely **architectural** and **transactional power estimation**. The architectural power estimation is the estimation of power by considering the full transactional characteristic of a network. In this case, we estimate the full static and dynamic power consumption for all NoC components. The transactional power estimation is the power consumption when an NoC is operated by a specific traffic pattern. In this case, we estimate full static power for all the

components and dynamic power for only those components that switch during simulation time. For example, during simulation time if the commuting of flits is half of the maximum NoC capacity, then the dynamic power consumption will be half of the maximum capacity of NoC.

2.10.6 Area

We are using two metrics for area in this thesis: router area and link area. Router area is the chip space that all the routers of an NoC occupy. It is measurable in terms of parameters of an NoC system. However, link area is different. Link length varies depending on the location of routers in an SoC system. The location of routers is assigned during the process of floor planning of a chip, which is not accessible during simulation time. To solve this problem, we assume an area for each core, and then by measuring the area of routers, we can calculate the NoC area without link area. Now, by having this area we can estimate the total link length of NoC system (see section 3.5).

2.11 NoC Simulation Systems Overview

The routing mechanism is one of the most important parts of NoC design. Many researchers have worked on different routing algorithms in 2D topologies. Ge-Ming Chiu presented the odd-even turn model for designing partially adaptive wormhole routing algorithms without adding virtual channels [19]. They concluded that the communication performance of 2D meshes may be improved under non-uniform traffics by using their proposed model. Wang Zhang et al. compared XY routing algorithm and Odd-Even routing algorithm for a 2D 3x3 mesh using NIRGAM simulator [20]. They concluded that Odd-Even routing algorithm performs better than XY routing with Constant Bit Rate traffic. Jie Wu proposed a simple and efficient deterministic fault-tolerant and deadlock-free routing for 2D meshes without virtual channels

[38]. The novelty of their approach is the use of two boundary lines at the east and west of a faulty block.

In terms of power modeling, two main power models ranging from RTL power estimation tools [5] to early stage architectural power models [6 and 7] have been proposed in NoC area. The first model requires complete RTL code and simulate slowly in the order of hours, while an architectural power model takes in the order of seconds. The shortcomings of the RTL power estimation models has led the development of early-stage architectural power models and simulators such as the widely-used ORION [6] used in academia and incorporated into industry tool chains for NoC power modeling.

The main methodology of the RTL-level NoC power models can be described by two papers [5, 39]. Banerjee et al. have developed an RTL level power model for NoCs by first extracting the SPICE level net list from the layout and then integrating the characterized values into the VHDL based RTL design [39]. An accurate power characterization of a range of NoC routers was performed through RTL synthesis and place and route using standard ASIC tool flow by Synopsys [5]. Both studies has limitations of RTL-level power simulation such as slow simulation time, and they require detailed RTL modeling that make them unsuitable for early-stage NoC design space explorations. Besides, power models cannot be targeted for future technology nodes. Patel *et al.* [40] first proposed an architecture level power model for interconnection networks, deriving its power estimates based on transistor count. As the model is not instantiated with architectural parameters, it cannot be used to explore tradeoffs in router micro architecture design. ORION is an early-stage architectural power model for NoCs that was originally proposed and released in 2002 [6]. It has since been widely used in academia and incorporated into industry tool chains (such as Intel, AMD, IBM, and Freescale)[6]. Bona *et al.*

also presented a methodology for automatically generating the energy models for on-chip communication infrastructure [40]. However, the focus is on bus based and crossbar based communication for SoC systems. Bhat *et al.* proposed an architecture level regression analysis model for different router components based on the energy volumes obtained from simulations using MAGMA tools [41].

In terms of NoC simulator development, a lot of research has been done on developing a SystemC simulation platform for analyzing the different aspects of an NoC system. Among the current SystemC simulators, NIRGAM and NOSTRUM are the prominent one.

NOSTRUM is a SystemC NoC simulator intending to develop NoC architecture [33]. It mainly concentrates on the communication issues from the general to the application level. NOSTRUM utilizes wormhole switching, regular topologies such as 2D mesh or torus, XY routing and deflecting routing. It employs a traffic flow generator that generates packets in spatial, temporal and size distributions.

NIRGAM is a SystemC based discrete event, cycle accurate simulator for NoC design [42]. It provides considerable support to examine different NoC designs in terms of routing algorithms and various topologies. It is capable of simulation on 2D regular mesh or torus topologies using wormhole switching. The routing algorithms are XY, OE, and SOURCE routing. The traffic generator in NIRGAM generates packets in constant and bursty bit rate.

FAANOS covers all the functions of NIRGAM and NOSTRUM works with some new approaches. It considers different bit rates in the application-oriented workloads and supports them by producing appropriate output results. FAANOS implements a complete and diverse routing mechanism in 2D topologies by using the mechanisms such as SOURCE, VIRTUAL-CIRCUIT, Odd-Even and LP routing. These mechanisms lead to flexibility and simulation

accurate results, and it is more supportive for users to provide efficient NoC design environment. FAANOS employs an analytical method to estimate the power and area of NoCs that is a fast and accurate methodology in NoC area. It produces different power and area characteristics such as static, dynamic, architectural and transactional power estimation as well as link and router area that completely describe the power and area behaviour of an NoC design.

2.12 Chapter Summary

We described the concept, architecture and configuration of an NoC as well as outlined and challenged the characteristic of NoC from the point of view of our simulator. In switching techniques, we introduced the most current switching methods for example the wormhole switching that is the most popular and well suited on chip [18] and employed it as the exclusive and prominent switching technique in our simulator. In terms of topology, we introduced regular and irregular topologies. The different kinds of traffic patterns have been introduced. In terms of power and area, we proposed an analytical method which dive in micro-architectural structure of NoC modules and extract the power models of each module using the structural characteristics of smallest component of a CMOS circuit (transistor). We broke down and defined the concept of performance to four characteristics such as throughput, latency, packet drop, and link utilization. The concept of power was also broke down to four characteristics such as static, dynamic, architectural and transactional power estimation, and then they were defined. At last, we briefly explained some previous works in terms of routing algorithm, power modeling and framework development.

Chapter 3

Structure of NoC Simulator

In NoC technology, there are a wide variety of NoC simulators, ranging from very simple to complex. Minimally, a network simulator may enable a user to define basic NoC elements such as network topology, router, link, and traffic pattern in the network. More complex simulator can allow users to specify everything like the current routing methods or power and area simulation for the NoC. Our NoC simulator, FAANOS, is a complex simulator coded in SystemC language representing a flexible and accurate NoC simulation framework [4]. It gets the configuration of a given NoC and provides accurate results related to power, area and performance of the NoC. We have used SystemC to create a cycle-accurate model of NoC hardware architecture, and NoC interfaces. The outline of this chapter is as follows. The infrastructure of NoC simulator is introduced in section 3.1. The hardware, power and area modeling of the NoC are described in sections 3.2, 3.3 and 3.4 respectively. An estimation methodology for the link length is proposed in section 3.5, and the performance, power and area estimations of NoC are provided in section 3.6. Finally in section 3.7, we propose an evaluation flow model for NoC systems.

3.1 Infrastructure of NoC Simulator

The block diagram of the structure of our simulator (FAANOS) is demonstrated in Fig. 3.1. The blocks and arrows indicate the flow of data and information in the simulator. The first executive block in the figure is the User Interface. It gets NoC parameters in the form of various data numbers and text file related to NoC cores and their parameters. The User Interface converts the input data into core graph and core switch graph of NoC for visualization, and finally it generates a file which will be used for NoC simulation in the next block. The NoC Simulation block consists of all the hardware descriptions of NoC modules in the form of SystemC and some of the SystemC modules are utilized for producing output results.

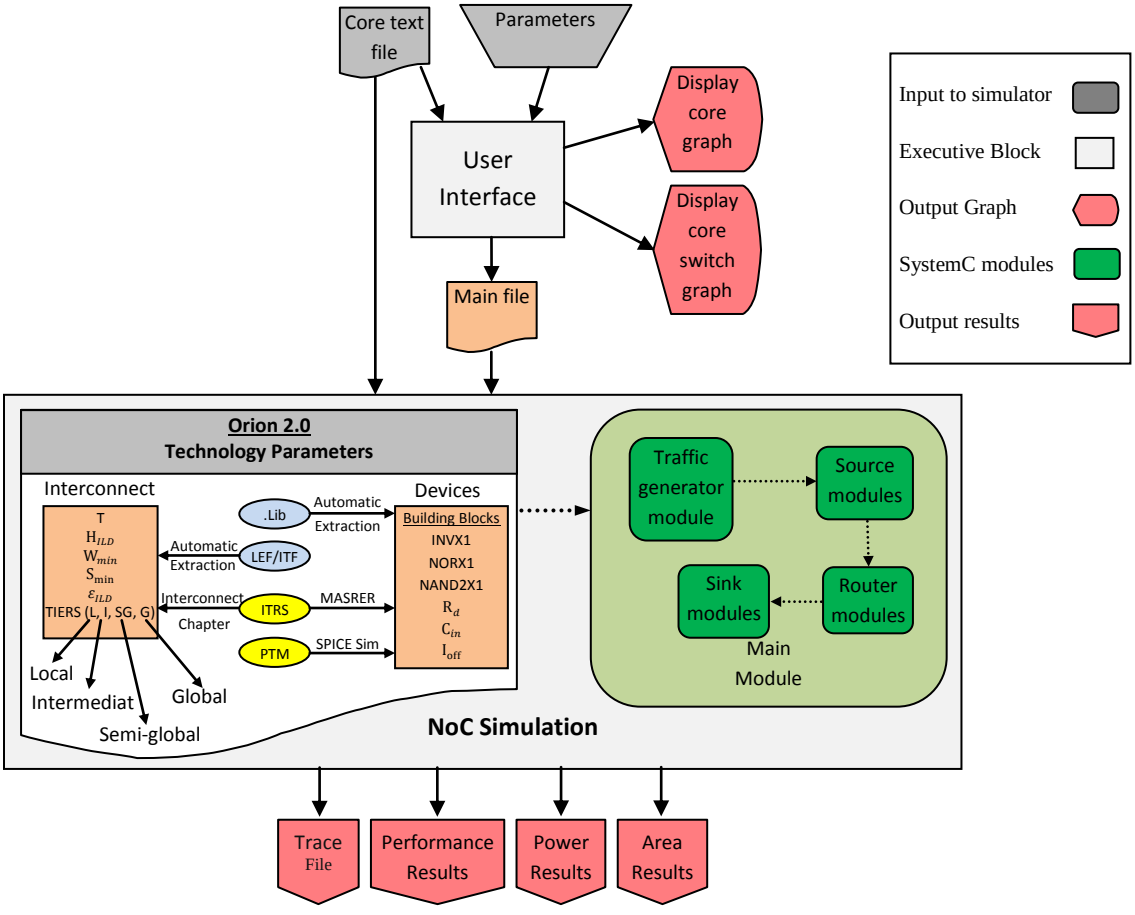


Figure 3.1: NoC Simulator Structure

The main functions of NoC Simulation block is completely depicted in the figure as described next. The inputs to this program are the files, which are prepared by the user and the User Interface program, as well as libraries of power and area parameters. The NoC power and chip area details are embedded in the simulator. The NoC Simulation block uses these inputs to synthesize the final NoC structure and then start the simulation process. After the completion of simulation, it produces the area, power and performance results of NoC accompanied by a trace file.

3.2 Hardware Modeling of NoC Simulator

Our simulator is divided into a number of NoC component modules that represent various areas of functionality of an NoC design. These modules are the basic container object* of SystemC. To better understand the structure of simulator, we start from a small NoC design that is depicted in Fig. 3.2. It consists of a source module, a traffic generator, a sink (receiver) module and a router module. These four modules are connected by communication links together. The source, sink and traffic generator modules play the role of SoC cores in the simulator. However, we use only one traffic generator. The source, sink and router module can be more than one and are identified by unique ID numbers. For the specific NoC of Fig. 3.2, the ID for the source, router and sink modules are the same and equal to zero.

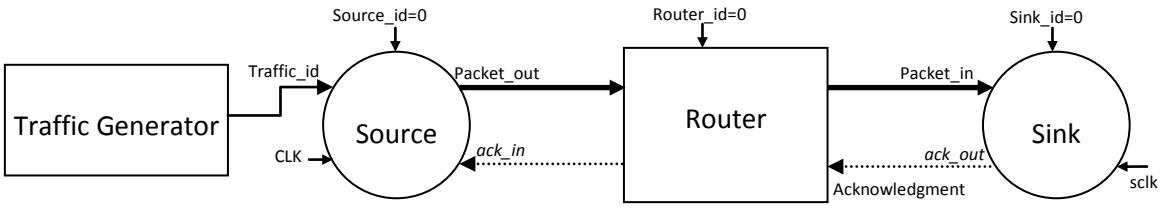


Figure 3.2: a small NoC

*A container is a class, a data structure, or an abstract data type .

Each module contains of two basic elements such as port and process [43]. Ports allow communication among the modules. Processes are the main computation elements which execute concurrently. In the following sections, we describe the port and process element for each module.

3.2.1 Modeling the Source Module

The source module and traffic generator play the role of a source core in our NoC simulator. The source module in is configured as a greedy source, which produces synthetic and random packets at the highest rate possible and at the earliest opportunity. The source module uses a particular message structure, which gives the design access to the packet definition and methods associated with the packet. A message consists of packets where a packet is formed by various number flits. A flit is the smallest element of data which travels inside NoC at a clock cycle. In FAANOS, a packet has at least two flits of header and payload. The header flits are needed to route data from the source node to the sink node. The header flit has various routing information as depicted in Fig. 3.3 and described as follows:

- **Source and sink address bits** are used to identify the sender and receiver nodes. The size of them is defined by a parameter such as *FW*. The parameter *FW* is determined depending on the number of cores in NoC. For example, if the number of cores is sixteen meaning *FW* should be more than four. It also determines the size of the FIFO buffer such as 2.*FW*+5.
- **Virtual channel bits** determine the number of virtual channels used in each channel of the router. The parameter *vcid* specifies the size of these bits. The default is 2 bits for 4 virtual channels.

- **Vertical/horizontal bit** determines the movement of packet. It is obvious that switching this bit switch XY to YX routing. In the router this bit can be changed depending on the routing algorithm used in the simulator.
- **Imaginary clock bit** flips between 0 and 1 in each new flit and plays the role of a clock in a packet. An NoC system is stimulated by events and an event is invoked for a new flit. If two flits have the same contents, then an event will not be created. In order to differentiate between two flits, this clock bit is employed in every flit.
- **Tail/Header bit** determines the end of a packet and this bit is set high in the last flit. The payload can be more than a flit. Each payload flit carries data as well as *tail/header* and *imaginary clock* bits.

A: header flit

B: payload flit

Figure 3.3: Header and payload flit

The source module has four input ports *traffic_id*, *source_id*, *ach_in* and *CLK* as well as one output ports *packet_out* (see Fig. 3.2). The output port, *packet_out* is connected to the router, and the source module uses it to send packets to the router. The input port, *source_id* has the identification code that identifies the source module in the NoC. The input port, *traffic_id* is connected to traffic generator and at each clock cycle has a destination address related to that source. The input port, *ach_in* is connected to router and gets the acknowledgement for sending a new packet. The input port *CLK* is connected to the clock generator. The source module has a

process, which is sensitive to positive edge transitions on the input port *CLK*. The main duty of the source process is to make packets depending on the packet specification and to send packets inside NoC depending on the characteristics of traffic pattern. The source process is invoked by a clock event. It calls two subroutines depending on the chosen traffic pattern as follows.

In the case of regular pattern (i.e. fixed, uniform and locality), the function *regular_gen()* will be executed. At the beginning of *regular_gen()* function, there are local variables to keep the frames and local data and the function continuously sends data packets to network and have a termination condition depending on how many packets has been sent. When the termination condition happens, the source module stops the simulation and starts the calculation of output results. The other condition is whether the router is ready to receive packets. In each clock, the acknowledgment port of router is read, and if the router is not ready, it increases the packet drop of the source and returns. When the router is ready, it goes into a loop to send packets. The function gets the destination ID from the input *traffic_id* and generates the header flit and sends it to the router. Then the function generates the flit consisting of data, *h_d* bit and *flit_clk* bit, and then sends it to the router. The pseudocode for the source process is illustrated in Fig. 3.4.

In the case of irregular pattern or application-oriented traffic, the function *irregular_gen()* is executed. In this case, the source module does not use the traffic generator module, however it reads a core txt file that contains the core information of NoC application and then saves this information in an array buffer. Then it derives the destination addresses and rate of packet related to the sources. As each source is independent from the other sources, a source should independently calculate some parameters to operate under the same configuration with other sources. The source process then scales all the packet rates by dividing it with the minimum packet rate. This scaling is useful for large packet rates. Therefore, we have considered a variable

SCALE_P in the configuration parameters that activate this operation in the simulator. The source process then calculates the parameter *PERIOD* from the scaled packet rates and assigns as simulation time. PERIOD is the minimum guaranteed time in which every source sends at least a packet to its destination. The simulation time is the same in all the sources and all the sources operate under the same scaling factor. The source process then enters to a loop, while has a termination condition depending on the number of packets sent. When termination condition happens, that stops the simulation as well as starts calculating the output results. Another condition in the loop is whether the router is ready to receive packets. In each clock cycle, the source process reads the acknowledgment port of router. If the router is not ready, it increases the dropped packet variable and returns. When the router is ready, it goes into a loop to send packets. The function depending on the destination addresses derived from core txt file generates the header flit as well as sends it to the router. Moreover, the function generates the flit consisting of the data bits, and then sends them to the router. The pseudocode of the source process is illustrated in Fig. 3.4 that can be consulted for further details.

```
void source:: func()
{
        if(IRREG_TRAFFIC==0)regular_gen();    //when traffic pattern is regular
        if(IRREG_TRAFFIC==1)irregular_gen();  //when traffic pattern is irregular
}

void source ::regular_gen()           //the source function using regular traffic pattern
{
  define the local variables;
  Initial the local variables;

  while( sim_count++ < SIM_NUM )      //continue till current time is less than SIM_NUM
  {
    wait();
    if( sim_count > (SIM_NUM*LOAD) ) goto exclude;   //when current time is higher than
                                                     //SIM_NUM*LOAD do nothing
    ack = (bool)ach_in.read();        //read  the acknowledgment port of the router
    if(ack) ++flt_drp;                //when  the router does not accept packet
    else                              //when  the router accepts packet
    {
        Handle the burst messages;
        Read the input port which connected to traffic generator;
        Save it as destination ID;
        Generate packet related to destination ID;
        Send packet;
        Keep records of the total time and number of packets
        Initial for next clock cycle
        exclude:;
    }
  }
    sc_stop();
}

void source ::irregular_gen()//the source function using irregular traffic pattern
{
        Define the local variables;
        Initial the local variables;
        Read file, core.txt, and save in a array buffer;
        Derive the associated destination Id;
        Derive the associated rate of packet;
        Derive the parameter PERIOD and assign to SIM_NUM;
        Scale the rate by dividing it by the minimum rate;
        New initiation of the local variables;

        while(sim_count++ <(SIM_NUM*period)) //continue till current time is less than SIM_NUM
        {
                wait();
                ack = (bool)ach_in.read();    //read  the acknowledgment port of the router
                if(ack) temp++;               //when  the router does not accept packet
                else                          //when  the router accepts packet
                {
                        Handle the burst messages;
                        Generate packet related to destination ID;
                        Send packet;
                        Keep records of the total time and number of packets;
                        Initial for next clock cycle;
                        exclude:;
                }
        }
    stop function
}
```

Figure 3.4: Pseudocode of source process

3.2.2 Traffic Generator

In FAANOS, the traffic generator is responsible for dictating the traffic pattern to the source modules. When the traffic is application-oriented, it is implemented as a separate function for each source module. We described it an *irregular_gen()*function as part of a source module in the previous section. But when the traffic is spatial (i.e. uniform or locality) and topology is regular (i.e. Mesh or Torus), it is synthesized as a single module.

In this configuration, by considering the traffic generator as a single model, we can fully control on the generation of uniform or locality traffic. For example, in each clock cycle, a new traffic pattern is generated once for each destination address. In other words, each source sends packet to a sink and each sink receives the packet from a source. The traffic generator module has a number of output ports that are connected via an array of signals to every source in the NoC. In each clock cycle, the traffic generator sends the address of destination of each source via these signals. It contains one process which runs at each positive edge of clock cycle. The traffic pattern has three types: fixed, uniform and locality. The process first checks the traffic configuration, and then depending on the pattern it activates a particular pattern as follows.

In the case of fixed traffic pattern, the traffic generator sends a fixed traffic pattern to the sources during simulation time. This fixed traffic pattern should be assigned in advance by the user.

In the case of uniform traffic pattern, the process randomly chooses a destination address from the current available destination addresses and sends to a source. In this pattern, each destination address is repeated one time and no source sends packet to itself. The pattern can be changed in each clock cycle.

In the case of locality traffic pattern, the locality coefficients are assigned in advance by the user and before starting the simulation. These coefficients determines with what probability a destination address can be chosen for a source. For example, if we assume that there are four coefficient probability in the system that they are intialized as *LOC_ONE*=8, *LOC_TWO*=4, *LOC_THREE*=3, *LOC_FOUR*=1, so for each source, the probability of choosing the destination among its neighbours is assigned for each source as follows. The probability of choosing a destination among the first neighbours far from the source is 8 out of 16. The probability of choosing a destination among the second neighbours far from the source is 4 out of 16. The probability of choosing a destination among the third neighbours far from the source is 3 out of 16. Finally the probability of choosing a destination among the fourth neighbours far from the source is 1 out of 16. Therefore, the process first considers a source and randomly chooses a number between 1 and 16. If the number is between 1 and 8, the destination is randomly chosen among the first neighbours far from the source. If the number is between 9 and 12, the destination is randomly chosen among the second neighbours far from the source. If the number is between 13 and 15, the destination is randomly chosen among the third neighbours far from the source. Finally, if the number is 16, the destination is randomly chosen among the 4th neighbours far from the source. The other consideration of making the pattern is that each destination address should be repeated one time in the pattern and no source sends packet to itself. The pattern is changed in each clock cycle. At the end, the destination address of each source is sent to the output port connected to the source. The pseudocode in Fig. 3.5 shows the operation of the traffic generator process.

```
// traffic_gen.cpp
void traffic_gen:: func()
{
   if(fixed)
   {
        Send the destination address of each source
        to the related output port depending on the fixed pattern;
   }
   if(UNIFORM)
   {
        Randomly chooses a destination address from
        the current available destination address;
        Generate traffic pattern;
        Send the destination address of each source
        to the related output port based on uniform pattern;
   }
   if(LOCALITY)
   {
        Consider a source;
        Randomly Choose a number among the sum of the locality coefficients;
        Determine the neighbours related to the chosen number;
        Randomly choose a destination address amonge the chosen neighboures;
        Generate traffic pattern;
        Send the destination address of each source
        to the related output port based on the locality pattern;

   }
}
```

Figure 3.5: Pseudocode of function in the traffic generator module

3.2.3 Modeling the Sink Module

The sink module accepts packets from the router module and keeps record of the number and time of incoming packets. It plays the role of a receiver core in the NoC. When the sink module successfully receives a packet, it sends an acknowledgment bit back to the router module. The sink module has four ports consisting of three input ports, *packet_in*, *sink_id* and *sclk*, and an output port, *ack_out* (see Fig. 3.2). The input port, *packet_in* accepts packets from the router. The clock port, *sclk* is connected to the clock generator. The input port, *sink_id* has a fixed value that identifies the sink module in the network. The output port, *ack_out* is used to send acknowledgment bit to the router. The sink module contains a process *receive_data* that is invoked when a new packet arrives on *packet_in* port (packet event) and a positive edge transitions on the clock port (clock event). In the case of a packet event, the process first stops receiving of new packet from the router. Then it reads packet and keeps the records of time and

number of incoming flits. In the case of a clock event, the process lets the router send new packet. The clock adjusts the speed of sink by controlling the acknowledgment to the router. The pseudocode in Fig. 3.6 shows the operation structure of a sink process.

```
void sink::receive_data()
{
   if ( clock event )
        {
               let the router send new packet;
        }
   if (packet event)
        {
               stop receiving new data;
               read data;
               keep records of the total time of received flits;
               keep records of the total incoming flits;
        }
}
```

Figure 3.6: Pseudocode of sink process

3.2.4 Modeling the Router Module

Two kinds of router namely the regular and irregular are used in our simulator. The regular router has five input ports and five output ports as shown in Fig. 3.7. It is used in the regular topologies such as Mesh or Torus. The regular router employs all the routing mechanisms described in chapter 2. However, the irregular router has flexible input and output ports depending on the application specific topology of the system. It is modeled to have maximum 16 input/output ports as well as it is used in irregular topologies.

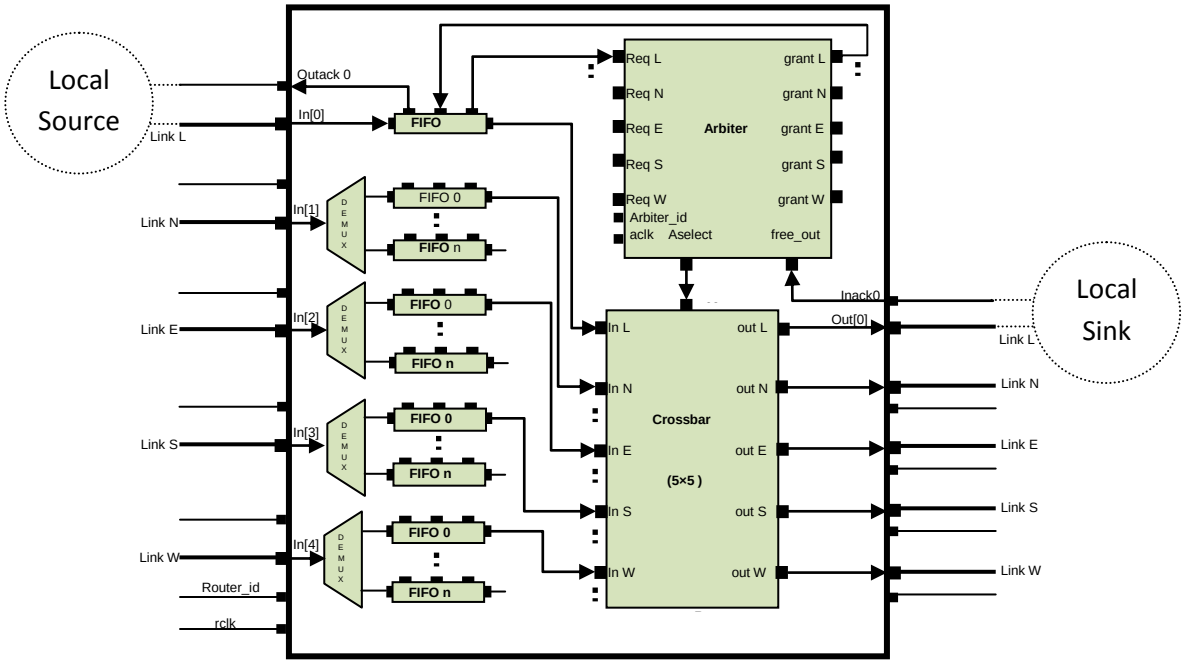


Figure 3.7: 5×5 wormhole router or regular router

The router module accepts packets from the source (or other router modules) and passes them to the sink (or other router modules). When the router receives a packet, it puts the packet into a channel. The address of the channel is determined by the incoming packet. The router checks whether the channel is full or no. If the channel is full, the router sends back an acknowledgment bit to the source module to tell the module to hold any further packets for the channel. The router also waits for acknowledgment from the receiver module after it sends a packet to the receiver. The router consists of some lower level modules such as *FIFO, crossbar, arbiter* and *demux* which are connected by signals together as illustrated in Fig. 3.7. A regular router module of 5×5 size contains 22 ports consisting of twelve input ports and ten output ports. The first input port, *in [0]* accepts packets from the source module and port *inact0* accepts acknowledgment bit from the sink module. Port *outact0* and *out [0]* send acknowledgment bit to the source module as well as data packets to the sink module respectively. The input port *router_id* has the constant value

of router ID. The clock input port *rclk* is used to get clock pulse from clock generator. Figure 3.7 shows these ports. The router process contains a process called *r_func ().* This process is sensitive to the events on the four input ports: *in [1], in [2], in [3]* and *in [4]*. When a new packet is received, the router function *r_func ()* is invoked to keep the records of the number of incoming packets. All the router tasks like incoming packets, acknowledgments, routing and transferring packets are done by the lower level modules of the router. The router only binds these modules and executes the router process. To provide a better understanding of how they work, we describe the journey of a header flit inside the router. Assume a local source module injects a header flit into the input port of first *FIFO* module. The *FIFO* module writes the flit into the tail of its buffers. When the flit emerges at the header of *FIFO* module, a request containing the route information is sent to the request port of arbiter module (*Req L*) for the desired output port (assume the north output port). The arbiter module performs the required arbitration. When the request is granted, the arbitration result is sent to the configure port of crossbar module (*config*). A grant signal is also sent to the *grant* port of *FIFO* module. Then the *FIFO* module activates its read port resulting the injection of flit to the input port of crossbar module. The flit then traverses through the crossbar module from its input port *In L* to its north output port *Out N*. Finally, the flit will leave the router. The following sections describe these modules and their implementation in detail.

a) Arbiter Module

When a router is the heart of NoC, the arbiter is the brain of router. The *arbiter* module handles all the methods in a router like the routing and switching algorithms. When the router has one virtual channel, the *arbiter* module has eight input ports and six output ports as shown in

Fig. 3.7**.** The request and grant ports are connected to FIFO buffers. *Aselect* port is connected to the *crossbar* module and when the arbitration is done, it will have the free requested output port. *Free_out* is connected to the *in_act* port of router and contains the acknowledgment from the receiver modules. *Arbiter_id* is connected to the *router_id* so that the arbiter has access to *id* of the router. *Aclk* is connected to *rclk* and by this means to the router clock generator. The arbiter module contains one process called *a_func ().* This process is sensitive to the events on the request ports and the positive edge of *aclk*. The arbiter process has one main function called *a_func ()* and eleven sub functions.

The arbiter process performs its jobs in two modes. In the first mode, it is invoked by a clock event and in the second mode by the request event. In other words, when a packet is injected into an input port of a router, it is directed by a demultiplexer into the first free virtual channel (FIFO buffer). The *FIFO* module sends the routing address of packet to the arbiter as a request event. When the arbiter detects that event, it reads the destination address and checks that whether the output address is free. If it is free, then the packet will be sent through that output port. The arbiter then disables a specific bit in the variable, *free_output* meaning that no data can be sent through the output port. This bit stays disable until the next clock event. When a clock event is invoked, the arbiter first checks that whether any output port gets free. If it is free, the arbiter enables the *free_output* bit related to that output port. Enabling this bit means that the related output port is ready to operate. The second duty of clock event is to pay attention to any unanswered requests. If there is any unanswered request, the arbiter checks its requested output port. If it is free, the data can go through that port. If the output port is not available, the request will stay until the next clock event. Figure 3.8 lists the pseudocode of arbiter process.

```
void arbiter :: a_func()
{
  while( true )  {
        wait();
        initiate the local variable in each while loop;
        if ( aclk.event()) {   //Start of clock event
          Start of whether any output port is free ;
          End   of whether any output port is free;
          Start of checking whether there is any remaining request for output port;
          Start of condition when input is reseved and after a while output will be freed;
           End of condition when input is reseved and after a while output will be opened;
           Start of condition when a header flit is in VC and requested output port is freed;
                Start line_probe algorithm;
                End   line_probe algorithm;
           End   of condition when a header flit is in VC and requested output is freed;
           Start of condition when input is reseved and after a while output will be freed;
           End   of condition when input is reseved and after a while output will be opened;
           Start of condition when a header flit is in VC and requested output is freed;
                Start line_probe algorithm;
                End   line_probe algorithm;
           End   of condition when a header flit is in VC and requested output port is freed;
          End of checking whether there is any remaining request for output port;
        End of clock event;
        Start of request events;
        else    {
        if (req0.event())      {
                Start of sending the body of packet;
                End   of sending the body of packet;
                Start of reading the header and sending the packet;
                       Start ODD_EVEN;
                       End   ODD_EVEN;
                       Start line_probe algorithm;
                       End   line_probe algorithm;
                End   of reading the header and sending the packet;
                Start for checking whether output port is free;
                End   for checking whether output port is free;
                Start of sending the body of packet;
                End   of sending the body of packet;
                Start of reading the header and sending the packet;
                       Start ODD_EVEN;
                       End   ODD_EVEN;
                       Start line_probe algorithm;
                       End   line_probe algorithm;
                End   of reading the header and sending the packet;
}}}     //End of the request events, while and a_func()
```

Figure 3.8: Pseudocode of arbiter process

b) Demultiplexer Module

The demultiplexer module receives packet from a sender module via the input port of the router. The module (*demux*) directs the packet to a FIFO module depending on the *vcid* value of header flit. When the router has four VCs, the *demux* module has one input port and four output port as illustrated in Fig. 3.9. It has one process called *d_func()*. The process *d_func()* is sensitive to event on the input port *d_in*. When an event happens, the function *d_func()* is provoked. First it read the data and if it is a header flit, the function stores its *vcid* value for a switch condition. In the switch condition, the process jumps to a case depending on the *vcid* value of the header flit. The case instructions write the flit to the input port of associated channel. Figure 3.10 depicts the pseudocode of the *demux* process.

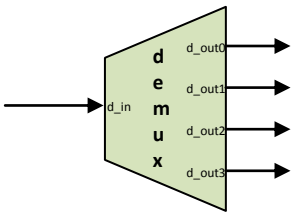


Figure 3.9: *demux* module

```
void demux :: d_func()
{
        while( true )
        {
                wait();
                read input flit;
                if the flit is header{ store its vcid to v_demux; }
                switch (v_demux)
                {
                        case 0: write flit to the output port of associated channel;
                        case 1: write flit to the output port of associated channel;
                        case 2: write flit to the output port of associated channel;
                        case 3:  write flit to the output port of associated channel;
}}}
```

Figure 3.10: Pseudocode of *demux* process

c) FIFO and Virtual Channel

Each input port of a regular router has at least a FIFO buffer module or maximum four *FIFO* modules. When the *demux* module directs a flit to the input port of FIFO module, the *FIFO* module writes the flit into the tail of its buffers. When the flit emerges at the head of *FIFO*, a request containing the route information is sent to the request port of arbiter module for the desired output port. After the arbiter module performs the required arbitration, it sends a grant signal to the *grant* port of *FIFO* module that leads to the activation of the read port of FIFO. The flit is injected to the input port of crossbar module. The detailed description of the above mentioned operation fallows.

The *FIFO* module has three input ports: *wr, grant* and *bclk* as well as three output ports: *re, req* and *ack* as illustrated in Fig. 3.11. It has a process called *f_func().* The process *f_func()* is sensitive to the events on the three input ports. The process function is provoked when one or more events happen on the input ports. In the write event *wr.event(),* the packet is stored in the tail of *FIFO* buffer. In the grant event *grant.event(),* the packet is sent to the crossbar module. In the clock event *bclk.event(),* the *FIFO* module sends a request to the arbiter module. The FIFO *struct* object provides a first in first out property to the buffers of *FIFO* module. The module creates this property by two functions i.e. *packet_out()* and *packet_in().* The *packet_in* function stores the flit in the tail of FIFO buffer and if the buffer is full, it causes the *FIFO* module to stop receiving new packets. The *packet_out* function shifts the contents of all registers once toward the head of *FIFO* module and if the module is full, it changes the condition in which FIFO starts receiving new packet. The pseudocode of Fig 3.12 depicts the functionality of this process.

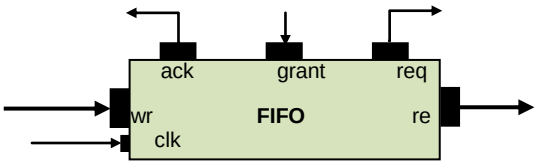


Figure 3.11: *FIFO* module

```
void buf_fifo :: f_func(){
        while( true ){
                wait();
                if (wr.event()){       // start reading incoming packets
                        read flit from input port;
                        store in the FIFO module;
                        send back the new condition of FIFO;
                        send a request to arbiter module;
                }
                if (grant.event()){    // start sending the packets out
                        send flit to crossbar module;
                        send back the new condition of FIFO;
                }
                if (bclk.event()){
                        send a request to arbiter module;
                }
}}
/* define FIFO */
struct fifo {
        define some registers (their number is equal to the size of FIFO);
        define a vaiable for the condition that FIFO is full;
        define a vaiable for the condition that FIFO is empety;
        define a vaiable (regnum) that points to the number of tail register;
        define two function packet_in and packet_out;
}
/* store data in the tail register of FIFO */
void fifo::packet_in(data){
        increament the variable regnum that points to the tail of FIFO;
        store data in the register that regnum points to it
        state the FIFO is not empty
        if (regnum == the size of FIFO) state the FIFO is full
}
/* read data from the header register of FIFO and shift the FIFO one register to the right */
packet fifo::packet_out(){
        decreament the variable regnum that points to the tail of FIFO;
        if (regnum == 0) state the FIFO is empty;
        else shift the content of FIFO one register to the right;
        state the FIFO is not full;
        send out the content of the first register;
}
```

Figure 3.12: Pseudocode of FIFO process

d) Crossbar Switch Module

When a flit is injected to the input port of crossbar module, the crossbar module reads the address of output port associated to the packet from the input port *config*, and then sends the packet out of the router via that output port. The functionality of crossbar is provided in detail here. When the router is regular and has four VCs, the crossbar module has eighteen input ports and five output ports as illustrated in Fig. 3.7. It has a process called *c_func ()*. The process *c_func()* is sensitive to the events on seventeen input ports (except *config* port). The process *c_func ()* is provoked when one or more events happen on the input ports. In the event, it reads the configuration address from the *config* port and then sends the packet via its associated output ports. The pseudocode of Fig. 3.13 is an event condition for the input *i0*. The remainder event conditions in the crossbar process are the same as the codes in this figure.

```
Read the output addresses from the config port
if (i0.event())
{
        Read the flit from the input i0;
        switch (associated output address)
                {
                case 2: send the flit of input port to the north output port;
                case 3: send the flit of input port to the east output port;
                case 4: send the flit of input port to the south output port;
                case 5: send the flit of input port to the west output port;
                break ;
        }
}
```

Figure 3.13: Pseudocode of a condition in crossbar process

3.2.5 Main Simulator Module

The *main* function is the top-level entity that ties all the NoC modules together and provides the clock generation and tracing capabilities. The pseudocode of *main* simulator module is shown in Fig 3.14.

```
// main_noc.cpp
# include "files"
int sc_main(int argc, char *argv[])
{
        Define local signals;
        Define local variables;
        Declare clocks;

        Instantiate the traffic generatore;
        Connect its ports to lacal signals;

        Instantiate the sources;
        Connect its ports to lacal signals;

        Instantiate the sinks;
        Connect its ports to lacal signals;

        Instantiate the routers;
        Connect its ports to lacal signals;

        Trace instractions;

        sc_start();            // start simulation

        Close trace files;     // stop simulaton

        if(REG_TRAFFIC)        // regular traffic
        {
                Calculate the performance, power and area metrics;
        }
        if(IRREG_TRAFFIC)      // irregular traffic
        {
                Calculate the performance, power and area metrics;
        }
}
```

Figure 3.14: Pseudocode of main function

The main SystemC function includes all of the modules in the NoC design. We instantiate each of the lower level modules as well as connect their ports with signals to create our NoC design. To instantiate a lower level module, the interface of the module must be visible. The local signals are declared to connect the module ports together. Three types of signals with different sizes such as *packet*, *Boolean* and *FW* (e.g. eight) are needed to cross connect the source, sink, traffic generator and router modules. After declaration of signals, there are three clock generation declarations: *clock1*, *clock2* and *clock3*. The number of clock generator is optional and can be equal to the number of modules in the design. However, we design the simulator to have three clock generators. *Clock1* generates clock cycle for the source modules. *Clock2* generates clock cycle for the router modules while *Clock3* generates clock cycle for the sink modules.

The modules in the simulator design are instantiated after the declaration statements. The traffic generator, source, sink and router module are instantiated as well as connected together with the locally declared signals. This completes the implementation of NoC simulator design. The SystemC program can now be built and run. To make it easier to determine if the design works as intended, we create a trace file with the built-in signal tracing methods in SystemC. After simulation is executed, we can examine the results stored in the trace file with a number of visualization tools that generate waveforms and tables of results. After the simulation is completed, the instructions related to the calculation of output results are executed. We explain the definition and calculation of these results in section 3.6. After the example is completely described in SystemC, the commands to build the simulator need to be specified.

3.3 Power Modeling of NoC Simulator

In this section, we propose an analytical method for estimating the power and area of NoC routers and interconnection links by varying the router architecture, ports, and interconnection link widths. We use the architectural models of different NoC components as well as the parameters and equations which are established inside these models to obtain the area and power consumption estimates. The results, which we obtain in this section, will be used not only to develop better energy/area models of NoC components, but also to understand the bottlenecks as well as suggest directions for the improvement of router architecture.

3.3.1 Derivation of Leakage Current (I_{leak})

As we mentioned in section 2.8.1, we need two characteristics I_{sub} and I_{gate} for each basic circuit components to estimate the static power. I_{sub} and I_{gate} for each component are listed in Table 3.1. They are derived from HSPICE and 65nm foundry SPICE model with corresponding technology parameters. Therefore, as the structure of each NoC module is hierarchically composed from these basic circuit components, FAANOS can calculate the I_{leak} for each based on the Equation 2.8 in section 2.8.

Table 3.1 I_{sub} and I_{gate} (per-micron of gate width) for each basic circuit component *i* at different input state *s*, 25°C and for high *Vth*.

i	s	I_{sub} (A)	I_{gate} (A)
NMOS	0	1.097e-07	4.622e-09
PMOS	1	3.172e-07	3.291e-09
INV	0	1.097e-07	4.622e-09
	1	3.172e-07	3.291e-09
NAND2	00	7.098e-08	3.549e-09
	01	1.134e-07	5.103e-09
	10	1.342e-07	1.194e-08
	11	1.766e-07	1.625e-08
NOR2	00	1.971e-07	6.701e-09
	01	1.034e-07	4.343e-09
	10	1.412e-07	8.048e-09
	11	7.245e-08	6.448e-09

3.3.2 Arbiter Leakage Modeling

In spite of different routing strategies in the arbiter, three types of arbiters namely matrix, round robin, and queuing are modeled in FAANOS. Here, we just explain the matrix arbiter, which is introduced in Orion [6]. For an arbiter with *R* requesting entities, one can represent its priorities by an *R*×*R* matrix. With one in row *i* and column *j*, if requester *i* has higher priority than another requester *j*, and zero otherwise. This method requires just R(R −1)/2 matrix elements in flip-flops. Equation 3.1 represents the relation between the *n*th grant and the requests.

$$grant_n = req_n \times \prod_{i<n}(\overline{req_i} + \overline{m_{in}}) \times \prod_{i>n}(\overline{req_i} + \overline{m_{in}}) \quad (3.1)$$

where

- req_i is the *i*th request;
- $grant_n$ is the *n*th grant;
- m_{ij} is the *i*th row and *j*th column element in the matrix.

Figure 3.15 shows the combinational logic of matrix arbiter. By considering Equation 3.1 and Fig. 3.15, the power static of matrix arbiter can be estimated by Equation 3.2.

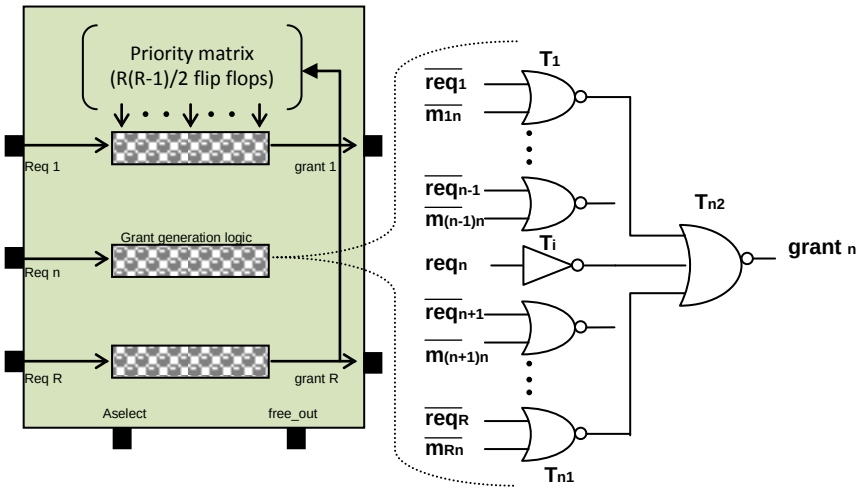


Figure 3.15: Gate level design of matrix arbiter

$$P_{leak}\ (\text{arbiter}) = I_{leak}\ (\text{arbiter}).\ V_{dd} \quad (3.2)$$

where

- I_{leak} (arbiter) = I_{leak} (NOR2). ((2R-1)R) + I_{leak}(INV). R + I_{leak} (DFF). R(R-1)/2
- R= number of request

3.3.3 Matrix Crossbar Leakage Modeling

FAANOS models the crossbar in two kinds: matrix and multi-tree. Here, we explain the matrix model of Crossbar. Figure 3.16 shows the gate level structure of normal matrix crossbar. The matrix crossbar consists of several horizontal input wires over vertical output wires, plus trans-gate or tri-state buffers at the cross-points. These tri-state buffers allow each input port to be electrically connected to any output port [44]. Table 3.2 shows the parameters and equations related to the static power of matrix crossbar.

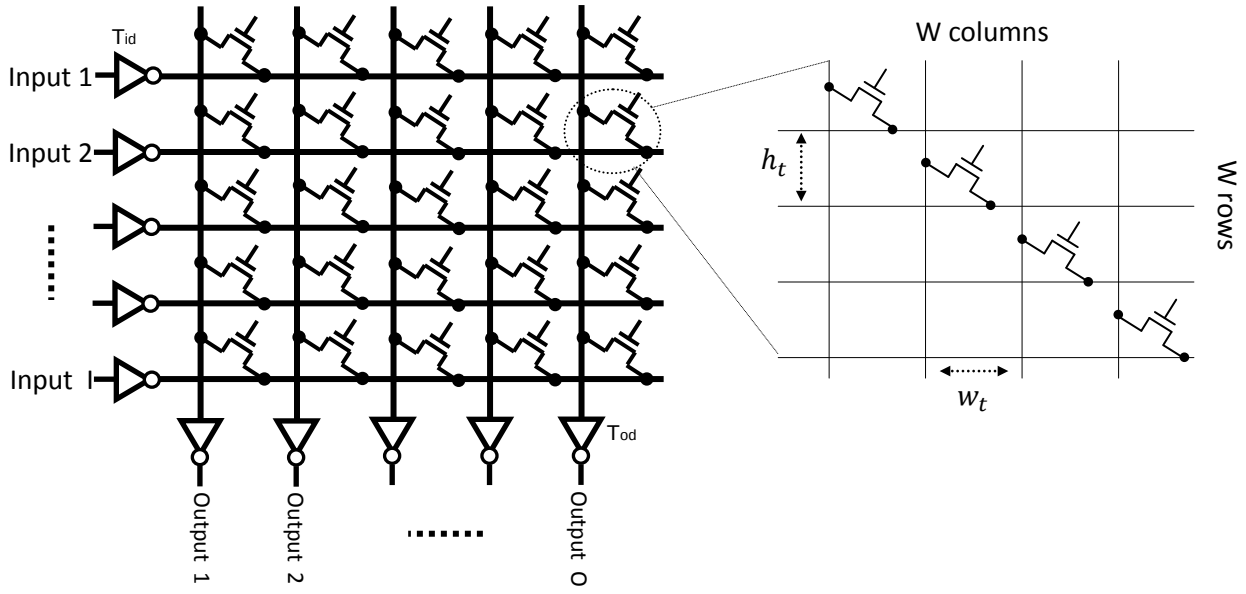


Figure 3.16: Matrix crossbar with *I* input ports, *O* output ports and W port width in bit. The connectors are transmission gate.

Table 3.2 Leak parameters of Matrix crossbar model		
I	No. of crossbar input ports (default 5)	
O	No. of crossbar output ports (default 5)	
W	Port width in bits (default 21)	
I_{leak} (transmission gate)	I_{leak}(NMOS). I .O. W	(3.3)
I_{leak} (input driver)	I_{leak}(INV). I . W	(3.4)
I_{leak} (output driver)	I_{leak}(INV). O. W	(3.5)
I_{leak} (control signal)	I_{leak}(INV). I .O	(3.6)
I_{leak} (crossbar)	I_{leak} (transmission gate)+ I_{leak} (input driver)+ I_{leak} (output driver)+ I_{leak} (control signal)	(3.7)
P_{leak} (crossbar)	I_{leak} (crossbar). V_{dd}	(3.8)

3.3.4 FIFO Buffer Leakage Modeling

The FIFO buffer is modeled in two types of SRAM and Register in FAANOS. Here, we explain the SRAM model of FIFO buffer. Figure 3.17 shows the gate level structure of normal SRAM. By considering this figure and the parameters and equations in Table 3.3, the static power of SRAM FIFO buffer model is estimated by Equation 3.16.

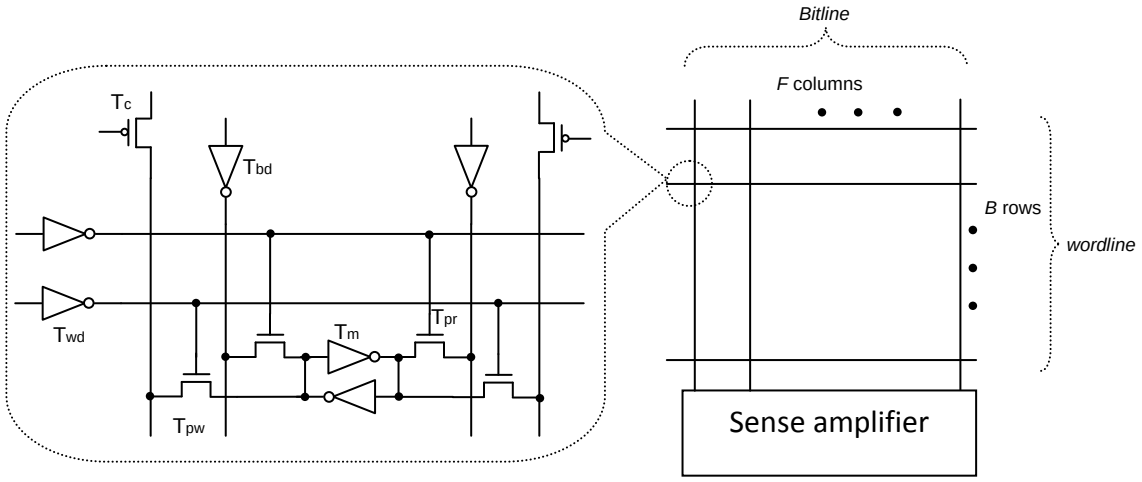


Figure 3.17: FIFO buffer with one read and one write port.

Table 3.3 Leak parameters of SRAM FIFO buffer model		
F	Flit width in bits (default 21)	
B	Buffer size in flits (default 2)	
P_r	No. of buffer read ports (default 1)	
P_w	No. of buffer write ports (default 1)	
V_CH	The number of virtual channel (default is 1)	
I_{leak} (bitline)	2. P_w .F.I_{leak}(INV)	(3.9)
I_{leak} (pre-bitline)	2. P_r .F.I_{leak}(PMOS)	(3.10)
I_{leak} (wordline)	(P_r+P_w).B. I_{leak}(INV)	(3.11)
I_{leak} (memory cell)	2.B.F. I_{leak}(INV)	(3.12)
I_{leak} (read port pass)	2. P_r .F.B.I_{leak}(NMOS)	(3.13)
I_{leak} (write port pass)	2. P_w .F.B.I_{leak}(NMOS)	(3.14)
I_{leak} (in FIFO buffer)	I_{leak} (bitline)+ I_{leak} (pre-bitline)+ I_{leak} (wordline)+ I_{leak} (memory cell)+ I_{leak} (read port pass) + I_{leak} (write port pass)	(3.15)
I_{leak} (FIFO buffer)	I. V_CH . I_{leak} (in FIFO buffer)	
P_{leak} (FIFO buffer)	I_{leak} (FIFO buffer).V_{dd}	(3.16)

3.3.5 Physical Link Leakage Modeling

The static power of links is due to the repeaters inserted in them. For repeaters, the leakage occurs in both output states. NMOS devices produce leakage current when the output state is high while PMOS devices leak when the output state is low. This is applicable to buffers also as the second stage devices are the primary contributors due to their large sizes. As mentioned, leakage power has two main components: sub-threshold leakage and gate-tunnelling current. Both components depend linearly on device size. Therefore, leakage power can be calculated using the equations given below [25]:

$$P_s=(pn_s+pp_s)/2 \quad (3.17)$$

$$pn_s=kn_0+kn_1.W_n \quad (3.18)$$

$$pp_s=kp_0+kp_1.W_p \quad (3.19)$$

Where

- pn_s and pp_s are the leakage power for NMOS and PMOS devices respectively;
- kn_0 = -6.034, kn_1= 26.561, kp_0 = 1.238 and kp_1 = 27.082 are coefficients determined using linear regression against 65nm low-power library.

State-dependent leakage modeling can also be performed using Equations 3.18 and 3.19 separately.

3.3.6 Component Modeling in Dynamic Power

This section explains the methodology of dynamic power modeling which we use in FAANOS. As we mentioned in section 2.8.2, the switching energy is calculated by Equation 3.20.

$$E = 0.5\alpha C V_{dd}^{\ 2} \tag{3.20}$$

Where

- α is the switching activity;
- C is the switching capacitance;
- V_{dd} is the supply voltage.

To model an NoC module, we divide the module into several *atomic* components, whose switching activities and capacitance can be independently calculated. A component is *atomic* if it only consists of one capacitor, or multiple capacitors that have identical switching behaviour. This property ensures that an atomic component can be represented by one capacitance. Usually an atomic component consists of several physically connected capacitors. For example, a wordline is physically connected to the gate ends of pass transistors connecting the wordline and memory cells. In this way, the wordline and the gate ends of pass transistors belong to one atomic component. There are cases where an atomic component consists of capacitors that are not physically connected. For example, the gate ends of NMOS and PMOS transistors of an inverter and their drain ends belong to one atomic component because they always switch together. The dynamic power modeling of an NoC module is thus developed as described here. First of all, we divide the NoC module into atomic components. Secondly, we derive capacitance equations for each atomic component. Thirdly, the possible operations (functions) of NoC

module is identified (e.g. read, write, update, and their arguments). These operations and their arguments are supposed to be accurately reported by the simulator. At the end, the equations that compute switching activities (i.e. energy) of each atomic component are derived from the operation arguments.

Component Power Modeling

Table 3.4 gives the capacitance notations used throughout the following sections. We use CACTI to compute the actual values of C_g, C_d and C_w [35]. Transistor sizes can be user-input parameters or automatically determined by Orion 2.0 [44] with a set of default values from CACTI and applied with scaling factors from WATTCH [7]. Sizes of the driver transistors, e.g. crossbar input drivers are computed according to their load capacitance. E_x denotes the energy dissipation per switch of component *x* and is implicitly defined when component capacitance Cx is defined. For each component, we first describe its regular structure in terms of architectural and technological parameters. We then proceed with a detailed analysis and derive parameterized capacitance equations by taking into account both the gate and wire capacitances. The capacitance equations are then combined to estimate switching activity to determine energy consumption per component operation.

Table 3.4 Capacitance notations

Notation	**Description**
Cg(T)	Gate capacitance of transistor/gate T
Cd(T)	Drain capacitance of transistor/gate T
Ca(T)	only applicable if *T* is an inverter. *Ca*(T)=*Cg*(T) + *Cd*(T)
Cw(*L*)	Capacitance of metal wire of length *L*
*C*in_cnt	Input node capacitance of a crossbar connector
*C*out_cnt	Output node capacitance of a crossbar connector
*C*ctr_cnt	Control node capacitance of a crossbar connector
*C*FF	Switch capacitance of a flip-flop
*C*FC	Clock capacitance of a flip-flop
E_x	$1/2C_xV_{dd}^2$ or $C_x V_{dd}^2$ depending on how to count switches, provided C_x is defined

Metal Wire Capacitance

The metal wiring is modeled using the values for *bitmetal* and *Cwordmetal* derived from CACTI 5.1 [35]. These values include an expected value for the area and sidewall capacitances to the substrate and other layers. We use these values for the capacitance estimation of bitlines and wordlines in the FIFO module. Moreover, they are used to model the capacitance of pre-decode lines, data / address bus, and other signals in the memory. Although the capacitance per unit length would be probably lower for many of these busses than for the bit lines and word lines, we have used the same value for the sake of simplicity.

Drain Cap of Transmission and Tri-state Gates

Figure 3.18 depicts the structure of transmission and tri-state gate respectively. The transmission gate is made by a single nMOS or pMOS transistor, or by the parallel combination of both.

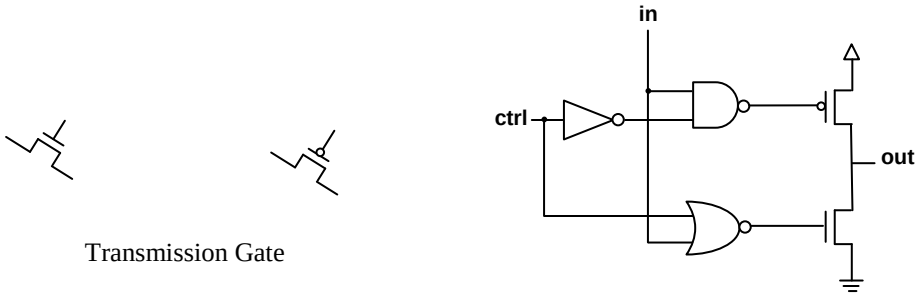


Figure 3.18: structure of transmission and tri-state gate

3.3.7 FIFO Dynamic Modeling

Most of the FIFO power is consumed when NoC is in the transaction condition. For NoCs, a FIFO buffer can be implemented by either static RAM (SRAM) or shift registers [25]. Designers typically implement FIFO buffers as SRAM arrays. Y. Hoskote et al. have concluded that the FIFO buffers consume up to 22% of the total router power [45]. An explanation of both the SRAM array-based model and shift register-based model is provided as following.

SRAM-Based FIFO Buffers

Several architectural- level SRAM array power models have been proposed [25]. These models are adopted and made more fine grained and also modified to take into account the specific router micro architecture. For instance, a FIFO buffer does not need a decoder. Figure 3.19 shows the structure of an SRAM-based FIFO buffer and Table 3.5 lists the model parameters and equations [6].

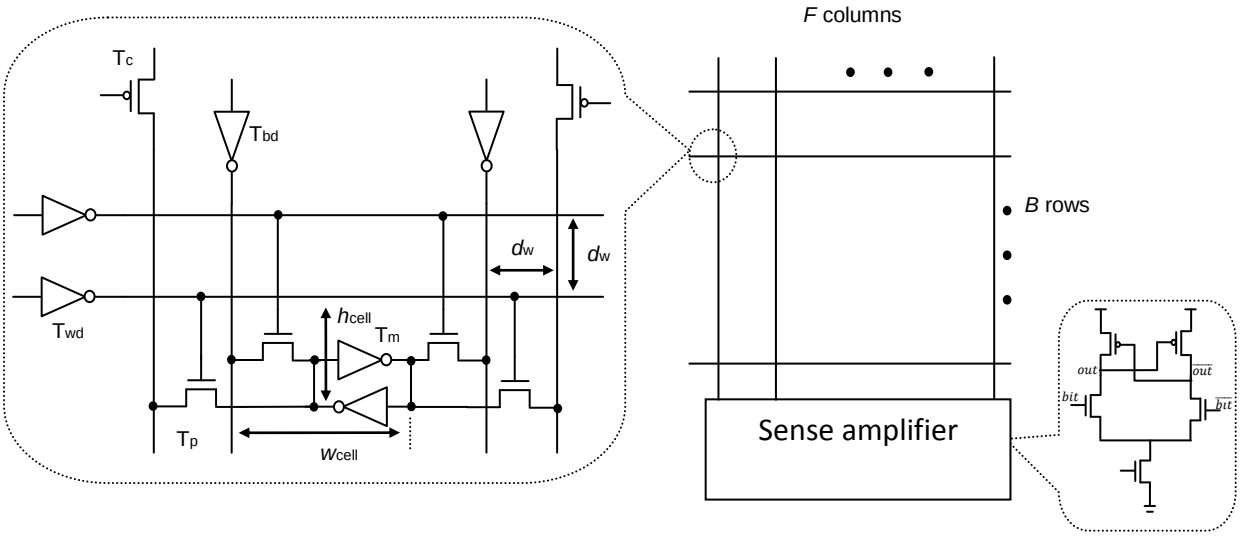


Figure 3.19: SRAM-based FIFO buffer with one read and one write port

Table 3.5 FIFO buffer model

Characteristic	Description	
Architectural parameters		
B	Buffer size in flits	
F	Flit size in bits	
Pr	No. of buffer read ports	
Pw	No. of buffer write ports	
Technology parameters		
Hcell	Memory cell height	
Wcell	Memory cell width	
dw	Wire spacing	
Model equations		
Word line length	Lwl = F[$w$cell+2(Pr + Pw)dw]	(3.21)
Bit line length	Lbl = B[$h$cell + (Pr +Pw)dw]	(3.22)
Read Word line capacitance	Cwl = 2FCg(Tp) + Ca(Twd) + Cw(Lwl)**	(3.23)
Write Word line capacitance	Cwl = 2FCg(Tp) + Ca(Twd) + Cw(Lwl)	(3.24)
Read bit line capacitance	Cbr = BCd(Tp) + Cd(Tc) + Cw(Lbl) +2Cg(Ts)	(3.25)
Write bit line capacitance	Cbw = BCd(Tp) + Ca(Tbd) + Cw(Lbl)	(3.26)
Pre charge capacitance	Cchg = Cg(Tc)	(3.27)
Memory cell capacitance	Ccell = 2(Pr + Pw)Cd(Tp) +2Ca(Tm)	(3.28)
Sense amplifier energy	Eamp from empirical model***	

**Tp is the pass transistor connecting bit lines and memory cells; Twd, the word line driver; Tbd, the write bit line driver; Tc, the read bit line precharge transistor; and Tm, the memory cell inverter.

*** V. Zyuban and P. Kogge, The Energy Complexity of Register Files, Proc. Int'l Symp. on Low Power Electronics and Design, 1998.

Register-Based FIFO Buffers

Some on-chip networks, such as the RAW microprocessor that evaluated by M. B. Taylor *et al,* use shift registers to form the FIFO buffer [46]. The flip-flops are used as the building block of shift registers [25]. Figure 3.20 shows the circuit schematic for a negative edge-triggered *D* flip-flop. It can be changed to a positive edge-triggered device by using the clock's complement. Hence, a FIFO buffer with size *n* can be implemented as a series of *n* flip-flops.

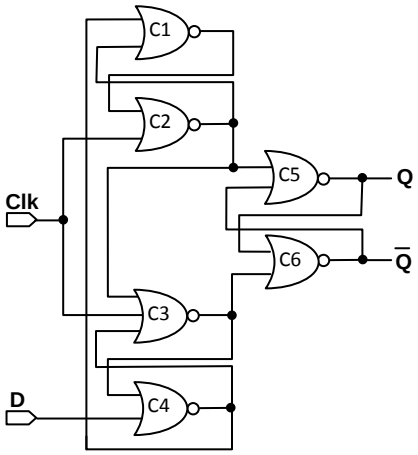


Figure 3.20: The schematic for a negative edge-triggered *D* flip-flop

Write Operation

The write operation occurs at the tail of shift register. Assuming the new flit isf_n and the old flit isf_o, the number of switched flip-flops is the Hamming distance between them. Therefore, the write energy is calculated by Equation 3.29.

$$E_{write} = H(f_n, f_o)\, E_{switch}^{ff} \tag{3.29}$$

Where

- $E_{switch}^{ff}$is the energy to switch one bit.

To simplify the analysis, let $\overline{H}$ denote the average switching activity (0.5 in the case of our simulator); then the average write energy can be defined by Equation 3.30.

$$\overline{E}_{write} = \overline{H} \,.\, E_{switch}^{ff} \tag{3.30}$$

Read operation

The read operation has two steps.

- The flit stored at the header of buffer is read into the crossbar. This step does not consume any energy in the buffer. Since the header of buffer is directly connected to the input port of crossbar.

- Subsequent flits in the buffer are shifted one position towards the header. If the buffer holds *n* flits before the read operation then (*n*-1) flip-flop writes are performed to shift the data. Hence, the average read energy is calculated by Equation 3.31.

$$\bar{E}_{read} = (n\text{-}1)\ \bar{E}_{write} \quad (3.31)$$

The capacitance value across different drive strengths is obtained from TSMC 65nm *G* and low-power standard cell library data sheets [25].

3.3.8 Crossbar Dynamic Modeling

We consider two ordinary crossbar implementations: multiplexer tree and matrix. Here, we explain just the matrix crossbar model. Figure 3.21 shows the model of a matrix crossbar. The data from an input port propagates to the input ends of connectors belonging to the same row. The open/close states of the connectors determine which output ports receive the input data. The components that can switch during data transfer are input, output, and control lines. Table 3.6 lists the model parameters and equations for the matrix crossbar.

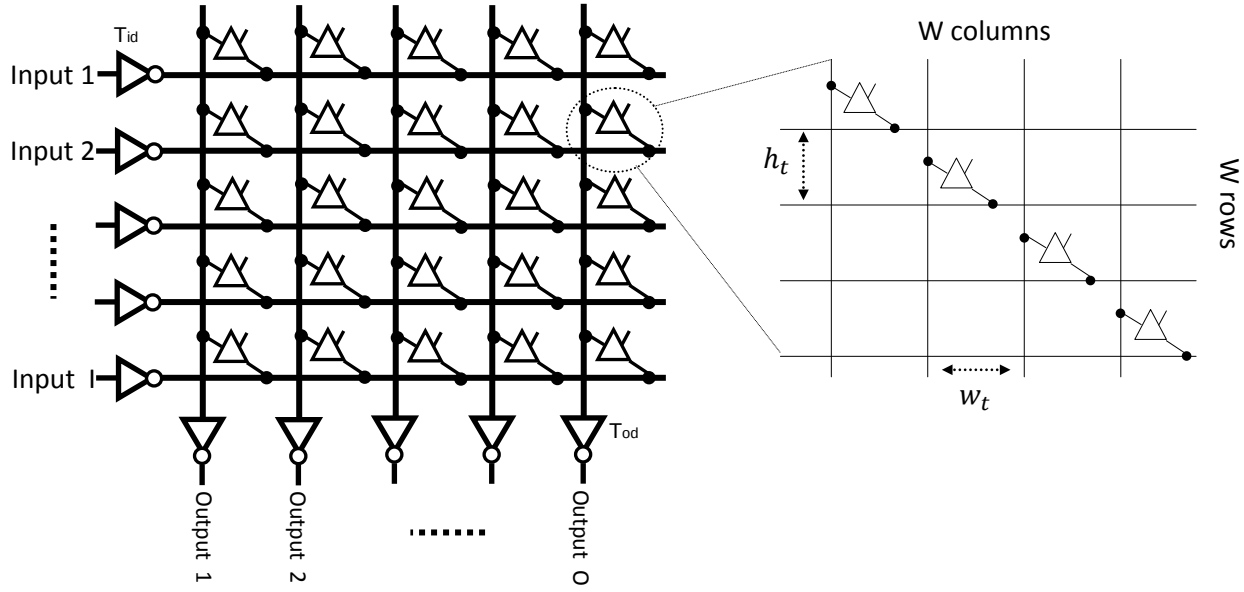


Figure 3.21: Matrix crossbar with *I* input ports, *O* output ports and W port width per bit. The connectors are tri-state buffers

Table 3.6 Matrix crossbar parameters and equations

Characteristic	**Description**	
Architectural parameters		
I	No. of crossbar input ports	
O	No. of crossbar output ports	
W	Port width in bits	
Technology parameters		
h_t	Track height	
w_t	Track width	
Model equations		
Input line length	$L_{in} = O.W.\ w_t$	(3.32)
Output line length	$L_{out} = I.\ W.\ h_t$	(3.33)
Input line capacitance	$C_{xb\_in} = O.C_{in\_cnt} + C_a(\text{Tid}) + C_w(L\text{in})$*	(3.34)
Output line capacitance	$C_{xb\_out} = I.C_{out\_cnt} + C_a(\text{Tod}) + C_w(L\text{out})$	(3.35)
Control line capacitance	$C_{xb\_ctr} = W.C_{ctr\_cnt} + C\text{w}(L\text{in}/2)$**	(3.36)

*Tid is the input driver, Tod is the output driver.
**We use average length for control lines and assume control lines are along the same direction as input lines.

3.3.9 Arbiter Dynamic Modeling

As mentioned earlier, there are three types of arbiters such as matrix, round robin, and queuing that are modeled in FAANOS. We will describe the matrix arbiter in this section. For an arbiter with R requesters, R(R −1)/2 flip-flops are required. The relation between the nth grant and the requests is given by Equation 3.1. Figure 3.22 shows the combinational logic of matrix arbiter. One can derive the capacitance equations of request, grant, and priority signals; internal nodes between the two levels of NOR gates; and the clock signal driving all the priority flip flops by using Equation 3.1 and Fig. 3.22 which are shown in Table 3.7.

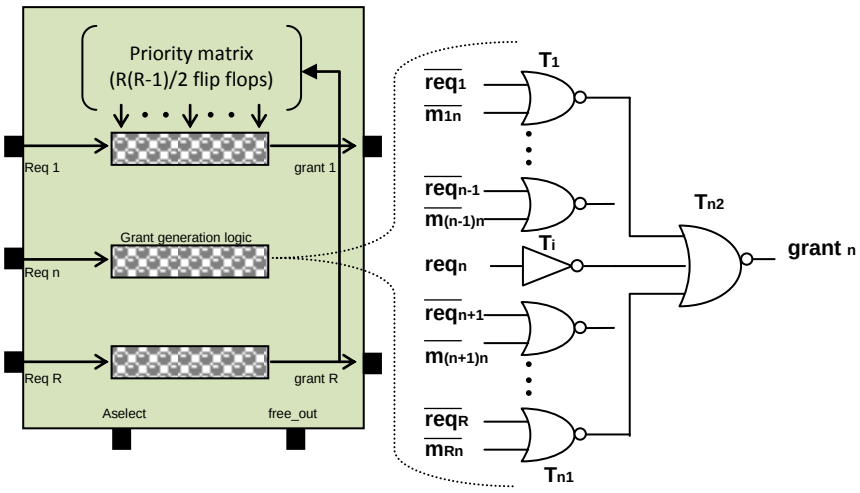


Figure 3.22: The gate level design of matrix arbiter

Table 3.7 Matrix arbiter model

Characteristic	**Description**
Architectural parameter	
R	No. of requesters
req_len	*request wire length estimation is ad hoc*
Model equations	
Request capacitance	$C\text{req} = C\text{a}(\text{Ti}) + (R\text{-}1)C\text{g}(\text{Tn1}) + C_w(\text{req\_len}) + C\text{g}(\text{Tn2})^*$ (3.37)
Grant capacitance	$C\text{gnt} = C\text{d}(\text{Tn2})$ (3.38)
Priority capacitance	$C\text{pri} = C\text{FF} + 2C\text{g}(\text{Tn1})$ (3.39)
Internal capacitance	$C\text{int} = C\text{d}(\text{Tn1}) + C\text{g}(\text{Tn2})$ (3.40)
Clock capacitance	$C\text{clk} = C\text{FC}$
* Tn1 is the first-level NOR gate; Tn2, the second-level NOR gate; and Ti, the inverter.	

3.3.10 Clock Dynamic Modeling

Clock distribution and generation include a major portion of power consumption in synchronous designs [47]. They represent a maximum of 33% of the power consumption in a high-performance router [45]. We estimate the term C_{clk} as shown in Equation 3.41.

$$C_{clk} = C_{sram\text{-}fifo} + C_{pipeline\text{-}registers} + C_{register\text{-}fifo} + C_{wiring} \quad (3.41)$$

Where

- $C_{sram\text{-}fifo}$, $C_{pipeline\text{-}registers}$, $C_{register\text{-}fifo}$, and C_{wiring} are capacitive loads due to SRAM-based FIFO structure, pipeline registers, shift register-based FIFO, and clock distribution wiring respectively.

We assume that all components are built by using static CMOS gates. Moreover, we assume an *H*-tree distribution style in the NoC design.

Clock in Memory Structures

We use the SRAM model of section 3.3.7 to determine the pre-charge circuitry capacitive load on the clock network. The circuit is just the pre-charging transistor (*Tc*), which is just a single ordinary PMOS transistor. Hence, its capacitance (C_{chg}) is due to its gate and drain end capacitances, *Cg* (of *Tc*) and *Cd* (of *Tc*) respectively as shown in Equation 3.42. In an SRAM FIFO with *B* buffers and flit size *F*, the total capacitance due to pre-charging circuitry can be derived using Equation 3.43, where *Pr* and *Pw* are the number of read and write ports respectively.

$$C_{chg} = Cg + Cd \tag{3.42}$$

$$C_{sram\_fifo} = (Pr + Pw)\ .F\ .B.C_{chg} \tag{3.43}$$

Clock in Pipeline Registers

Typical interconnection network routers have different pipeline stages. To advance, each flit must proceed through the steps such as routing computation, VC allocation, switch allocation, and switch traversal. We assume D flip-flop as the building block of pipeline registers. In a router with flit size of *F* bits and pipeline stages $N_{pipeline}$, the capacitive load on the clock due to pipeline registers is demonstrated as following formula.

$$C_{pipeline\_registers} = N_{pipeline}. F. Cff \quad (3.44)$$

Where

- *Cff* is the flip-flop capacitance and it is extracted from 65nm high-performance and low-power libraries.

Clock in Register-Based FIFOs

FIFO buffers can be implemented as a series of flip-flops. We assume simple D flip-flop to construct the FIFO. In a register-based FIFO with size *B* and flit size of *F* bits, the capacitive load on the clock can be computed by Equation 3.45 as given below.

$$C_{register\_fifo} = F .B. Cff \quad (3.45)$$

Where

- *Cf f* is the flip-flop capacitance and is obtained from TSMC 65nm *G* and low-power standard cell library data sheets [40].

Architectural parameters change the effective loading of each gate in the design. Therefore, to employ appropriate drive strength for the registers, we use their load capacitance and timing requirements. In this work, we assume minimum-size D flip-flops for all the registers.

Clock in Wiring Load

For a 5-level *H*-tree clock distribution, total wire capacitance is given in Equation 3.46.

$$C_{wiring} = (16/2 \times D+1\times 8/2\times D+2\times 4/2\times D+4\times 2/2\times D+8\times 1/2\times D) \times C_{int} \quad (3.46)$$

where

- C_{int} is the per-unit-length wire capacitance;
- *D* is the chip dimension.

3.3.11 Physical Link Dynamic Modeling

The main source of dynamic power of links is due to charging and discharging of capacitive loads (wire and input capacitance of next stage repeater). Internal power dissipation, due to charging/discharging of internal capacitors and short-circuit power, is only significant for repeaters when the input slew times are extremely large. Link power is a major component of the router total power (around 17%) [45]. Previous works, such as Heo and Asanovic, only use delay as the objective for buffer insertion [48]. These results in large repeaters increase the power consumption significantly. In this work, we use a hybrid buffering solution that minimizes the linear combination of delay and power. We evaluate the objective function exhaustively for given number and size of repeaters, while searching for the optimal (number, size) values. Link dynamic power is given by the equations as below.

$$P_{link} = \alpha.Cl \,.\, V_{dd}^2.\, fclk \tag{3.47}$$

$$Cl = Cin + Cgnd + Ccc \tag{3.48}$$

Where

- P_{link}, α, Cl , V_{dd} and $fclk$ denote the link dynamic power, activity factor, load capacitance, supply voltage, and frequency, respectively.

The load capacitance is the sum of input capacitance of next repeater *Cin*, and the ground (*Cgnd*) and coupling (*Ccc*) capacitances of wire being driven. *Cin* can be reliably obtained from the industry library files. The unit *Cgnd* and *Ccc* are also extracted from industry LEF files [25].

3.3.12 Approaches to Obtain Parameters

Detailed technology parameters are not always available for a given device. When these are unavailable, we propose to obtain them by using three approaches as follows.

- Measure the parameters from the floor plan or its photo if available.

- Determine the size of some transistors by using their load capacitance and timing requirements.
- Use default values, many of which come from the WATTCH framework [7].

Therefore, an NoC designer can use our model before determining all the low-level details. But knowing these technology parameters can greatly improve the model's accuracy.

3.4 Area Modeling of NoC Simulator

To estimate the router area we basically compute the area of each NoC module and sum them up with an additional of 10% (rule of thumb) of area to account for global whitespace. For each NoC module we first identify the implementation style of module and then decompose the module into its basic logical elements (e.g., gate-level netlist). Then, we use the gate area model described in section 2.9 to estimate the area of entire block.

3.4.1 Router Area

Designers typically implement buffers as SRAM arrays. Some NoC networks used shift registers due to less demanding buffer space. We have modeled the area of both implementations, but explain only the SRAM-based model here. We use the same SRAM-based FIFO buffer as in ORION [6]. Equation 3.49 and 3.50 compute the word line and bit line lengths of FIFO, respectively.

$$L_{word\text{-}line}=F.(W_{cell}+2(P_r+P_w)d_w) \quad (3.49)$$

$$L_{bit\text{-}line}=B.(H_{cell}+(P_r+P_w)d_w) \quad (3.50)$$

Where

- F, B, W_{cell}, H_{cell}, d_w, P_r, and P_w are flit size in bits, FIFO size in flits, memory cell width, memory cell height, wire spacing, number of read ports and number of write ports respectively.

Therefore, the total area for a FIFO with B buffers and flit size of F is calculated by Equation 3.51. In this model, H_{cell} and W_{cell} are computed using the gate area model described earlier in section 2.9.

$$\text{Area}_{\text{FIFO}} = \text{L}_{\text{word-line}} \,.\, \text{L}_{\text{bit-line}} \tag{3.51}$$

For other router components, namely, crossbar and arbiter we first decompose them into their circuit building block (i.e., gate level netlist). Then, using the gate area model we estimate the area of individual circuit components and compute the area of entire module.

3.4.2 Link Area

The area occupied by NoC links is due to wires and repeaters as illustrated in Fig. 3.23. We use the earlier gate area model described to estimate the area of repeaters. The area of global wiring can be calculated as given in Equation 3.52.

$$Area_{link} = F \,.(W_w + W_s) + sw \tag{3.52}$$

Where

- $Area_{link}$ denotes the wire area;
- F is the flit size in bits;
- sw is the repeater area;
- W_w and W_s are the wire width and spacing computed from the width and spacing of layer (global or intermediate) on which the wire is routed, and from the design style.

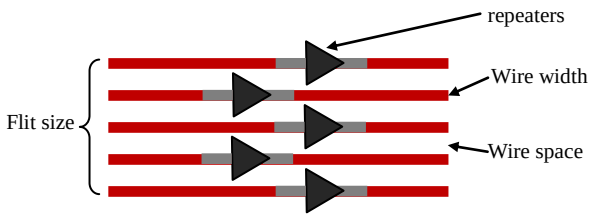


Figure 3.23: Layout model of wires and repeaters

3.5 Link Length Estimation

In order to estimate area and power dissipation of an NoC, we need to know the exact size and position of each core and the length of every link. These parameters are determined in the process of making chip specifically during the floor planning of system. We propose an estimation method which only needs the core areas of an SoC application to guess the maximum length of links on an NoC to reduce the complexity of NoC simulator and to avoid floor planning tools. We describe our method for 2D as well as application specific topologies, covering all the topologies used in FAANOS.

3.5.1 Maximum Link Length in 2D Topologies

An example of an SoC which is mapped on 2D topology, Mesh or Torus is illustrated in Fig. 3.24. The SoC is floor planned on an area with size AREA with N_CLM ×N_ROW cores. We assume all routers are located on the area as is shown in Fig. 3.24. In this condition, we need maximum length link equal to 2*(N_CLM + N_ROW)* $\sqrt{AREA}$ to route all the cores bidirectionally inside the chip for Mesh topology and two times of that of Torus topology. Equation 3.53 and 3.54 shows the average link length (*Ave_Link*) in Mesh and Torus topology respectively.

Ave-Link=2*(N_CLM+N_ROW)*sqrt(AREA)/(4*(N_CLM-2)*(N_ROW-2)+
3*2*((N_CLM-2)+ (N_ROW-2))+ 4*2) (3.53)

Ave-Link=2*2*(N_CLM + N_ROW)*sqrt(AREA) /(2*2*N_CLM * N_ROW) (3.54)

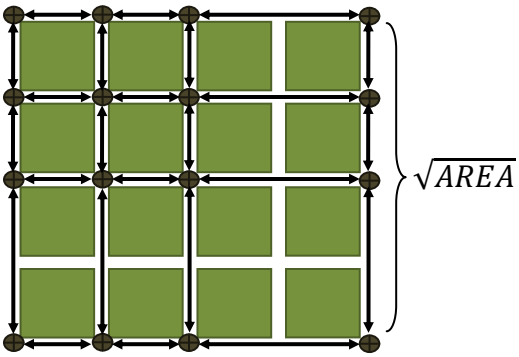


Figure 3.24: An example of 2D NoC floorplanning

3.5.2 Maximum Link Length for Application Specific Topologies

Figure 3.25 shows an SoC consisting of *N* cores, which are floor planned in an area with size AREA. We assume the SoC is of square shape and its NoC is mapped to an irregular topology. The condition which needs the longest link to floor plan the system happens when the NoC is placed in the corner of chip and each core is connected directly to it (Fig. 3.25). In this case, the maximum link is equal to the diameter of chip ($2\times\sqrt{\text{AREA}}$ in a 2D layout). As the cores are uniformly spread in the chip, the average of a link is half of the diameter of area ($\sqrt{\text{AREA}}$). Therefore, we need maximum length link equal to $N\times\sqrt{\text{AREA}}$ to route all the cores bi-directionally inside the chip. Equation 3.55 shows average of link length in an application specific topology.

$$\text{Ave-Link} = \sqrt{\text{AREA}} \tag{3.55}$$

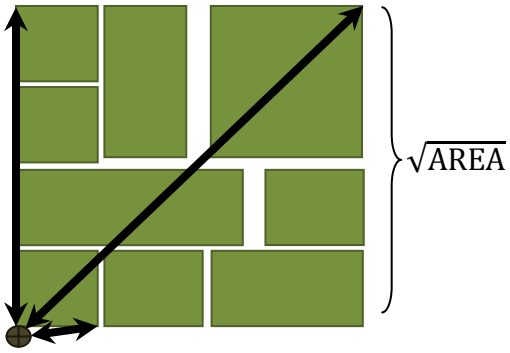


Figure 3.25: An example of application specific NoC floorplanning

3.6 Performance, Power and Area Estimation of NoC

We have described the performance, power and area metrics of each NoC module separately. In this section, we explain how these metrics are integrated and create the performance, power and area metrics of the NoC system.

3.6.1 Performance Estimation

The definition of performance characteristics have been presented in section 2.10. Now it is time to have an exact view of how these characteristics are measured and create the performance results in our NoC simulator. Figure 3.26 depicts the packet injection model in FAANOS. When data packets commute between the NoC modules, each module stores the NoC transaction characteristics in different variable memories. These variables facilitate the collection of transaction statistics and assist the integration of performance metrics in FAANOS. The following variables assist the integration of performance results in FAANOS

M	the number of network nodes
C	the number of link in the network
T	the number of simulation cycles
LOAD	a proportion of SIM_NUM in which the traffic generator produces packets
BURST	the number of packet generated in the burst time
PAUSE	the pause per flit between two burst messages
FF	the number of flits in each packet
N_{in}	the number of flits injected into the network
N_{of}	the number of flits offered into the network
Nout	the number of flits ejected from the network
N_i	the number of flits ejected from the source i
D_i	the links a flit i travels
T_{of}	the time a flit i offered into the network
T_{out}	the time a flit i ejected from the network

D_{of} or $\sum_{i=0}^{N_{of}} D_i$ the total link traveled by all offered flits

T_{of} or $\sum_{i=0}^{N_{of}} T_{of}$ the total time of flit offered into the network

T_{out} or $\sum_{i=0}^{N_{of}} T_{out}$ the total time of flit ejected from the network

The following formulas represent the general relationship between above terms and calculate the integration of performance results in FAANOS.

$$N_{in} = M \times (LOAD \times T) \times \frac{BURST \times FF}{(BURST \times FF) + PAUSE} \quad (3.56)$$

$$\text{Total Packet drop} = \frac{N_{in} - N_{of}}{FF} \quad (3.57)$$

$$\text{Packet drop in source i} = \frac{T - N_i}{FF} \quad (3.58)$$

$$\text{Link\_utilization} = \frac{D_{of}}{C \times T} \quad (3.59)$$

$$\text{Flit\_injection\_rate} = \frac{N_{in}}{M \times T} \quad (3.60)$$

$$\text{Throughput} = \frac{N_{out}}{M \times T} \quad (3.61)$$

$$\text{Latency} = T_{out} - T_{of} + (N_{of} - N_{out}) \times T \quad (3.62)$$

Where

- $M = K^2$ and $C = 4K(K-1)$ For a K× K mesh;
- $M = K^2$ and $C = 4K^2$ For a K×K torus.

In the latency calculation, we assume the flits that are injected but still not ejected the same as the flits that are ejected.

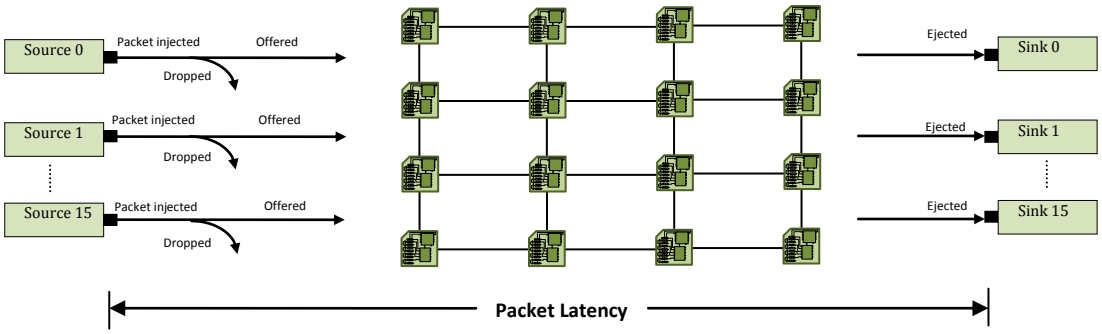


Figure 3.26: Packet injection model

3.6.2 Architectural Power Estimation

Static power consumption is a function of leakage current and leakage current is constant during transaction level of NoC system as explained in section 2.10.5. However, dynamic power consumption is different. For architectural power estimation, the estimating of power by considering the maximum transactional characteristic of network is considered. In other words, the maximum static and dynamic power consumption of all NoC components is estimated. In this section, we demonstrate the estimation of maximum dynamic power consumption by using a probabilistic approach. Maximum power, P_{max} is the power consumed with maximum achievable switching activity at a certain network load. To simplify the illustration, we assume a simple wormhole router as shown in Fig. 3.27. It has five input/output ports, where each input port has a buffer space for four 32-bit flits (B = 4, F = 32, Pr = 1 and Pw = 1). The router has a 32-bit wide, 5×5 crossbar (I = 5, O= 5 and W= 32) and a 4:1 arbiter at each output port. Assuming flits do not make u-turns (R = 4). The maximum power, P_{max} is calculated by Equation 3.63.

$$P_{max} = f_{clk}(5E_{buffer} + E_{crossbar} + 5E_{arbiter} + 5E_{vc\_al}) \quad (3.63)$$

where

- E_{buffer}, $E_{crossbar}$, E_{vc_al} and $E_{arbiter}$ are the energy of each input buffer, crossbar, VC allocator and arbiter consumes in one cycle.

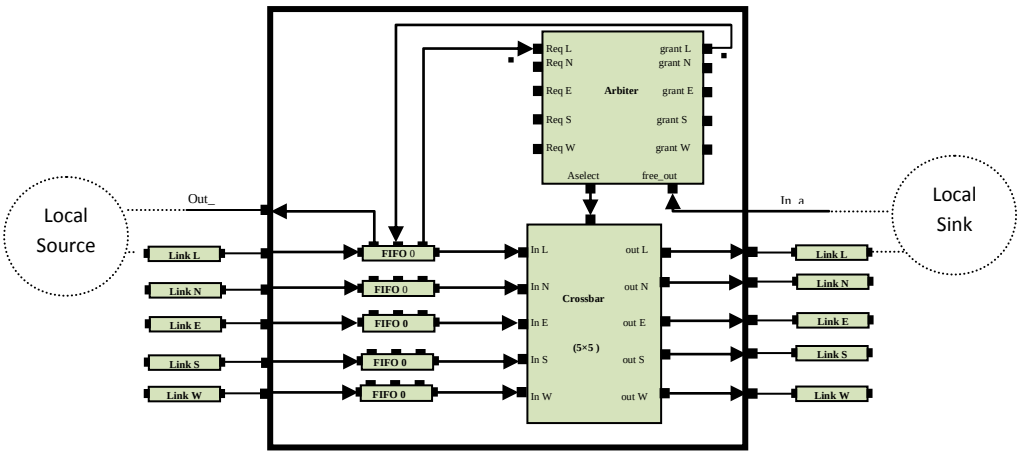


Figure 3.27: 5×5 wormhole router or regular router

We assume network traffic is uniformly distributed across all the router ports and define flit arriving rate, P_f as the probability that each input port receives a flit in every cycle. We also assume input traffic equals output traffic. Therefore, in one cycle, P_f flits are written into each input buffer and P_f flits are read out from each input buffer. The maximum switching activity is achieved when all 32-bit lines and memory cells switch during read/write.

$$E_{buffer} = P_f (E_{write} + E_{read}) \quad (3.64)$$

$$E_{write} = E_{wl} + 32(E_{bw} + E_{cell}) \quad (3.65)$$

$$E_{read} = E_{wl} + 32(E_{br} + 2E_{chg} + E_{amp}) \quad (3.66)$$

Assuming $5 \times P_f$ flits enter the crossbar in each cycle. A maximum switching activity happens when all $5 \times P_f$ flits travel to different ports. For example, there is no contention, and for each input/output port, all 32 lines will switch.

$$E_{crossbar} = 5.P_f .32(E_{xb\_in} + E_{xb\_out}) \quad (3.67)$$

The arbiter energy consists of the energy consumed by the arbitration action, and the energy consumed during clocking of flip-flops. Assume *L is* the number of flits in each packet. Since the contention and arbitration is only necessary for header flits therefore each arbiter receives one request in every L/P_f cycle. Upper bound of maximum switching activity comes from the situation when all the relevant priorities and all the internal nodes switch. Although in practice, this is not achievable. Clocking energy is summed for all the flip-flops and it is a constant.

$$E_{arbiter} = P_f / L \; E_{arbitration} + E_{clock} \quad (3.68)$$

$$E_{arbitration} = (4 -1)E_{pri} + 4(4 -1)E_{int} + E_{req} + (E_{gnt} + E_{xb\_ctr}) \quad (3.69)$$

$$E_{clock} = 1/2 \; .4(4 -1)E_{clk} \quad (3.70)$$

We add E_{xb_ctr} to E_{gnt} because grant signals load crossbar control signals. Similarly, we can derive the switching activity for average power. Since our power model is activity sensitive, it is also capable of computing average power, or more realistic power estimation based on the real data trace.

3.6.3 Transactional Power Estimation

The transactional power estimation is the power consumption when an NoC is operated by a specific traffic pattern as discussed in section 2.10.5. In this case, we estimate maximum static power for all the components and dynamic power for only those components that switch during simulation. When data travel between NoC modules, each module keeps records of its transactional characteristics. These records facilitate the collection of transaction statistics and assist the integration of power results in FAANOS. When a simulation is stopped, the traversal energy (the energy consumed at each transaction in a module) related to each module is

multiplied by the transactional record of that module and then accumulated and result in the energy consumption of NoC design.

Here, we describe the journey of a flit from one router to another to show how the transactional power consumption is estimated. Consider a router that has five input/output ports, a 5×5 crossbar and an arbiter as shown in Fig. 3.28. Each input port has two FIFO buffer module containing four registers (VC=2, FIFO=4), and each register has 32-bit. We also assume XY routing for simplicity. In the figure, we have added a virtual channel allocator (*VC_allocator*) module to separate the duty of *demux* module to a switch module (i.e. *demux)* and an allocator module (i.e. *VC_allocator*). The local source module injects a header flit into the local link. The header flit passes the local link and will be entered into the input port of router. The router module keeps a record of incoming flit in a variable called *v_link*. Then the flit will enter to *demux* module. The *demux* module keeps a record of incoming flit in a variable called *vc_al*. The *demux* module sends it to a FIFO module depending on the *vcid* value of header flit. The FIFO module writes the flit into the tail of its buffers and keeps a record of incoming flit in a variable called *v_wr*. When the flit emerges at the header of FIFO module, a request containing the route information is sent to the request port of arbiter module (*Req L*) for the desired output port, the north output port is assumed here. The arbiter module performs the required arbitration and keeps a record of request in a variable called *v_arb*. If the request is granted, the arbitration result is sent to the configure port of crossbar module (*config*). A grant signal is also sent to the *grant* port of FIFO module, leading to the FIFO read port be activated. At this stage, a record of grant signal is kept in a variable called *v_ read*. The flit is injected to the input port of crossbar module. The crossbar module keeps a record of incoming flit in a variable called *v_crossbar*.

The flit next traverses through the crossbar, from its input port (*In L*), to its north output port (*Out N*). Finally, the flit leaves the router and enters to link N.

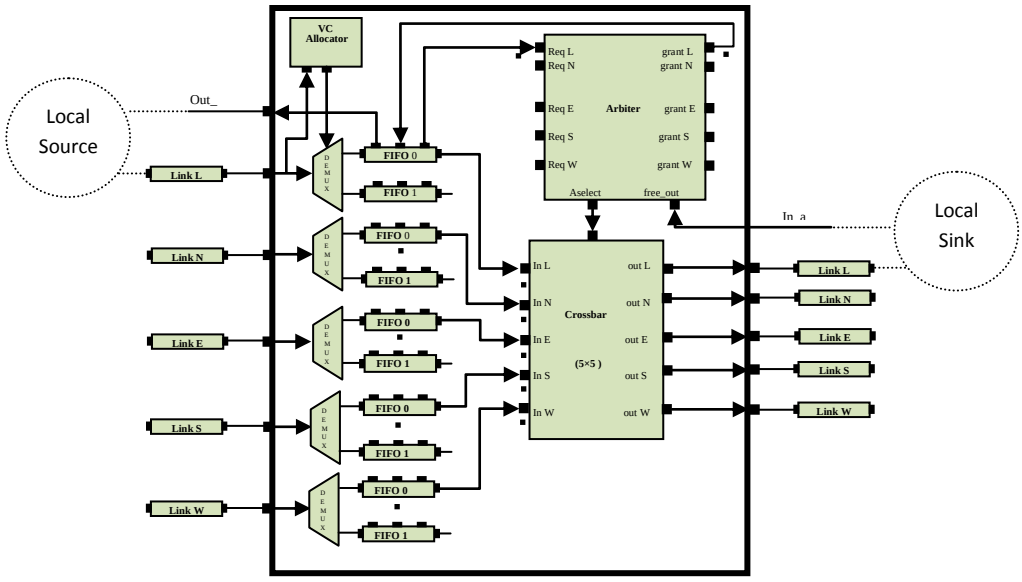


Figure 3.28: 5×5 wormhole router or regular router

By entrance of each flit to a router, the router keeps a record of incoming flit in *v_link*. In this way, a router covers all the links events except the links connected to the local sinks. Therefore, a router should also keep a record of incoming flit only for the exit link connected to a sink. Now, the dynamic energy for passing header flits inside a router during simulation can have the following equation.

$$E_{dyn\_router} = v\_link \,.\, E_{link} + vc\_al \,.\, E_{vc\_al} + v\_wr \,.\, E_{write} + v\_read \,.\, E_{read} + v\_arb \,.\, E_{arbiter} + v\_crossbar \,.\, E_{crossbar} \quad (3.71)$$

Where

- E_{link}, E_{vc_al}, $E_{arbiter}$ and $E_{crossbar}$ are the traversal energy of *link, VC_allocator, arbiter and crossbar* modules respectively when a transaction happens in a module.

- E_{write} *and* E_{read} are the traversal energies when a write and read transaction happens in a FIFO module.

As the arbitration and VC allocation happen only for header flits, so the dynamic energy for passing body flits inside a router during simulation is calculated by Equation 3.72.

$$E_{dyn\_router} = v\_link \cdot E_{link} + v\_wr \cdot E_{write} + v\_read \cdot E_{read} + v\_crossbar \cdot E_{crossbar} \quad (3.72)$$

Each router in the NoC accumulates the dynamic energy for the flits, which has passed through it. The following equations show the process of calculating the total power of NoC network, P_{network}.

$$\mathrm{E_{dyn}} = \sum_{i=1}^{m \times n} E_{dyn\_router}(i)$$

$$\mathrm{P_{dyn}} = \frac{\mathrm{E_{dyn}}}{N} \times f_{clk}$$

$$\mathrm{P_{static}} = (\mathrm{P_{leak}}\ (\text{VC allocator}) + \mathrm{P_{leak}}\ (\text{FIFO}) + \mathrm{P_{leak}}\ (\text{arbiter}) + \mathrm{P_{leak}}\ (\text{crossbar}) + \mathrm{P_{leak}}\ (\text{Link})) \times m \times n$$

$$\mathrm{P_{network}} = \mathrm{P_{dyn}} + \mathrm{P_{static}} \quad (3.73)$$

where

- $m \times n$, N, f_{clk} and E_{dyn} are the number of routers, the number of simulation cycle, the NoC frequency, the total dynamic energy of network during simulation time respectively;
- P_{dyn} and P_{static} are the total dynamic and static power of NoC network;
- P_{leak} *(VC allocator),* P_{leak} *(FIFO),* P_{leak} *(arbiter),* P_{leak} *(crossbar)*and P_{leak} *(Link)* are the static power of *VC_allocator, arbiter and crossbar* modules respectively.

3.7 An Evaluation Flow Model

We propose an evaluation flow model by which the effectiveness of an NoC system is evaluated in terms of performance, power and area metrics by using NoC simulator. In another aspect, the evaluation flow model is the process of configuring, implementing, recording,

analyzing, and iterating tasks by using FAANOS to gain the best effective results of an NoC system.

One should first configure the network and traffic pattern and then apply different traffic patterns to evaluate an NoC where Figure 3.29 shows this flowchart. The NoC evaluation flow may be iterative until a user is satisfied. There are two loop processes in the flowchart. The first one is partially implemented in the User Interface program and the rest in the NoC Simulator. In this step, users should specify the configuration of NoC system in terms of size and kind of message, topology, switching and routing mechanism, FIFO buffer and virtual channel. These configurations are specified in terms of NoC parameters and a core txt file. The parameters are requested via the console screen by the User Interface program that a user should include into the program. The core txt file is a file produced from the core graph of NoC application. The output of User Interface is a file that is used as the main file of NoC Simulation program (Fig. 3.1). The NoC Simulation program implements the hardware simulation part of FAANOS. It generates the performance, area and power results of NoC. In the second loop, the user cannot change the topology of NoC, but other configurations can be changed. These configurations are saved in the SystemC file, *main_noc.cpp*. The user can change them and run the NoC Simulation program iteratively to gain the best results of NoC. This evaluation flow method typically is called try and error method.

3.8 Conclusion

The structure of FAANOS has been analyzed in this chapter. We used SystemC, which is a C++ class library and dedicated language, to create the cycle-accurate models of the hardware part of simulator and a user interface program. The hardware modeling of every NoC module was

described in detail. We demonstrated the developing of our NoC simulator from a simple network to a more complex and automatic NoC simulator. In terms of power and area, we have presented an analytical model for the power and area of different modules of an NoC such as FIFO buffer, arbiter, crossbar, and link. We have introduced the different architectural model of each NoC module and then decomposed them into gate level and presented the technological and analytical parameters accompanied by the related formula. We proposed an estimation method for the average link length of NoC. We also described how the performance, power and area results of an NoC calculated in FAANOS. At the end, an evaluation flow model for early design of NoC has been presented.

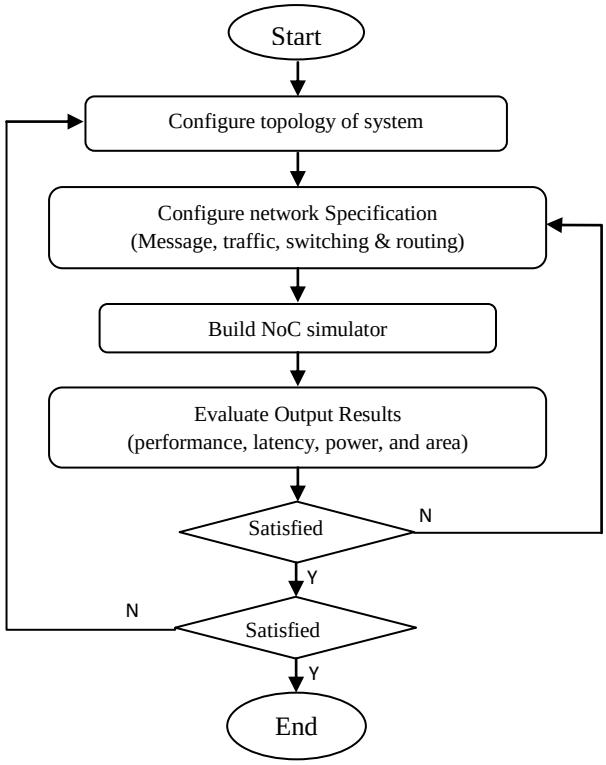


Figure 3.29: NoC evaluation flow

Chapter 4

Experimental Work

The routing mechanism refers as a deterministic or adaptive routing algorithm for the NoC. It is one of the most important network components of NoC design. The adaptive routing mechanisms are effective for regular topologies such as Mesh or Torus. A regular topology has a predictable and well-ordered shape where the routing paths can be addressed by well defined functions. These routing mechanism has become a good research subject for additional investigation and development. This has encouraged us to configure the simulator and implement the first set of experiment for different NoC routing mechanisms to investigate their performance trade-off. We show how an adaptive routing mechanism has some advantages over the deterministic one, and present a comparative analysis among LP routing and other routing techniques. For the second set of experiments, we investigate different SoC applications by trading power and area characteristics with the performance of NoCs. We illustrate different aspects of our simulator in the following sections. The configuration and installation of FAANOS is described in section 4.1. In section 4.2, a special type of contention is proposed for NoCs. The configuration for the first set of experiment is presented in section 4.3. Different experiments in uniform as well as locality traffic patterns are presented in sections 4.4 and 4.5.

The Mesh topology and application-oriented traffic have been experimented and their results are presented in sections 4.6 and 4.7. We present the second set of experiment by also taking into account the power consumption and chip area of NoC in sections 4.8. A summary of the chapter is given in section 4.9.

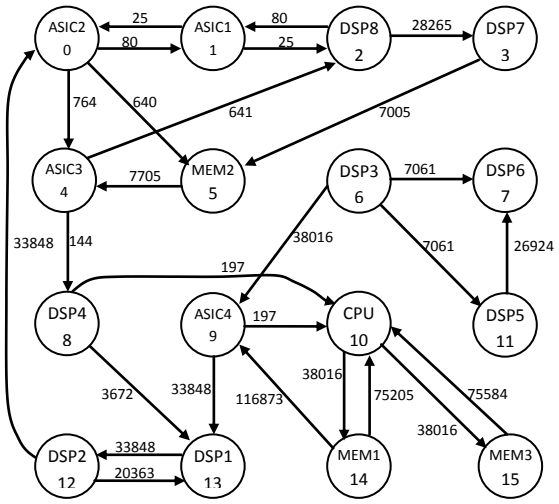
4.1 Configuration and Installation

FAANOS is a Windows base executable NoC CAD tool, which needs an (input) file, *core.txt* that is provided by the user from the specification of NoC application. The execution of FAANOS generates the User Interface program. After generating the User Interface program, a *main_noc.cpp* file is created. This file and *core.txt* construct the NoC Simulation program. A user can create a SystemC project in Microsoft Visual C++ and add these two files to it then build the project and execute it. After the completion of execution, the results of NoC performance, power and area are displayed on the console.

4.1.1 NoC Core Graph to Core Text File Conversion

A core graph is a special type of block diagram of an NoC application where the core blocks are connected by arrows indicating the data flow relationships among the core blocks. The core graph is a visual representation therefore we need to convert a core graph into a text file readable by the NoC simulator. We convert the core graph into a text file (*core.txt*) and utilize it as an input to our simulator. Figure 4.1 depicts an example of relation between the contexts of *core.txt* file with the core graph of an AV application. Each line of the *core.txt* file corresponds to an arrow between two cores of the core graph. Each line represents the direction of packet flow from source to destination. If the link between two cores has two arrows then two lines (in

the core text file) represent it. The first part of each line of the *core.txt* file represents the character “S” followed by and the source ID. The second part represents the character “D” followed by the destination ID. The third part is the character “R” followed by the rate of packet flow from the source to destination. For simplicity, the rate less than one should be rounded to one and the floating point numbers should be rounded to first natural number greater than or equal to them. When the rates of packet flow are close together, the floating part of numbers has prominent effect on the system. In this case, the user should scale up all the bit rates by multiplying each rate to 10, 100, etc to enable an accurate measurement. The simulator can also allow the *core.txt* file without the rate part where the rates are same and equal to the clock cycle of sources. For source by source distribution where each source has different temporal and message size packet flow, the *core.txt* file (data format) is extended. Each line gets longer by adding four more parts. The fourth part is the Burst information character “B” followed by the number of packets being generated continuously in the burst message. The fifth part is the pause in burst messages character “P” followed by the pause per flit between two burst messages. The sixth part is the flits character “F” followed by the number of flits in each packet. The seventh part is the number of bits character “N” followed by the number of bits in each flit. For example, the line *S2D5R150B5P2F4N8* indicates that source *2* sends packet to sink *5* at the rate of 150 packets per second. The source continuously sends five packets then pauses for two flits and does it consecutively. Each packet consists of four flits, and each flit is made of eight bits.



(a) AV benchmark core graph

```
S0D1R80
S0D4R764
S0D5R640
S1D0R25
S1D2R25
S2D1R80
S2D3R28265
S3D5R7005
S4D2R641
S4D8R144
S5D4R7705
S6D7R7061
S6D9R38016
S6D11R7061
S8D10R197
S8D13R3672
S9D10R197
S9D13R33848
S10D14R38016
S10D15R38016
S11D7R26924
S12D0R33848
S12D13R20363
S13D12R33848
S14D9R116873
S14D10R75205
S15D10R75584
```

(b) The context of *Core.txt*

Figure 4.1: core graph and core text file

4.1.2 User Interface

To lower the complexity of our simulator and to create a visual environment to communicate with the application, a User Interface program is provided in addition to the NoC Simulation program. The graphical User Interface allows the user to easily visualize the simulation environment. The text-based characteristic of User Interface may provide a less intuitive interface, but it permits various advanced forms of customization. User Interface is separated from NoC Simulation program because it makes a file that forms the main body of NoC Simulation program. We have created the NoC simulator program as an executable file that can be easily executed in Windows environment. The program starts by asking for the type of topology. One of the irregular or regular topologies should be selected. Then it asks different parameter values related to regular or irregular topologies. In the case of regular topology, the parameters are listed in the Table 4.1 as given below.

Table 4.1 Parameters and their descriptions in regular topology		
Parameter	value	Description
Regular/Irregular	1 0	Regular topology. (default is 1) Irregular topology.
Area	R	The area of SoC chip in mm^2. (default is 7)
N_COL	1≤N≤80	The number of columns in mesh or torus topology. (default is 4)
N_ROW	1≤N≤80	The number of rows in mesh or torus topology. (default is 4)
SIM_NUM	1≤N	The total clock cycle that simulator runs. In the case of irregular traffic this number is a coefficient of PERIOD, so it is better to be chosen less than 10. (default is 100)
LOAD	0<R≤1	A proportion of SIM_NUM which the traffic generator produces packets. This parameter is used only in uniform or locality traffic. (default is 1)
FW	2≤N	It is related to the address of source or destination. The flit width is equal to (2×FW+5) bits. Increasing it will extend the size of register and therefore will raise the resource cost. (default is 8)
NUM_FLIT	1≤N≤256	The number of flits per packet. (default is 5)
FIFO	1≤N≤256	The number of registers in FIFO. Increasing will extend the depth of FIFO and therefore raises resource cost. (default is 2)
TORUS	B	Torus topology. (default is 1)
MESH	B	Mesh topology. (default is 0)
XY	B	XY routing. (default is 1)
YX	B	YX routing. (default is 0)
ODD_EVEN	B	Odd_even routing, this routing is used only in mesh topology. (default is 0)
L_PROBE	B	L_probe routing, this routing normally employ one VC. In default L_probe uses XY. It also can use YX. (default is 0)
V_CH	1≤N≤4	The number of virtual channel. (default is 1)
IRREG_TRAFFIC	B	Implementing irregular topology as part of a regular topology. In this case users should provide a core text file. (default is 0)
FIXED	B	Fixed traffic. (default is 0)
UNIFORM	B	Uniform traffic. (default is 1)
LOCALITY	B	Locality traffic, in this case following LOC parameters should be specified. (default is 0)
LOC_ONE	N	Probability of sending packet to first place close to source. (default is 12)
LOC_TWO	N	Probability of sending packet to second place close to source. (default is 2)
LOC_THREE	N	Probability of sending packet to third place close to source. (default is 2)
LOC_FOUR	N	Probability of sending packet to fourth place close to source. (default is 0)

LOC_FIVE	N	Probability of sending packet to fifth place close to source. (default is 0)
LOC_SIX	N	Probability of sending packet to sixth place close to source. (default is 0)
BURST	1≤N	The number of packets generated continuously during the burst time. (default is 1)
PAUSE	N	The pause per clock cycle between two burst messages. (default is 0)
GAP_P	0<R≤1	A proportion of PERIOD added to itself and makes new PERIOD. (default is 0)
PRN_OUT	B	Show sending and receiving flits on the console screen. (default is 0)
R_ROBIN	B	Implementing the round robin technique in the routers. (default is 0)
SCALE_P	0<R≤1	The coefficient to scale PERIOD (0= maximum scale meaning calculating PERIOD based on the current rating packets, 1= no scale meaning calculating PERIOD based on the current rating packet divided by minimum rating packet in the system). (default is 1)

In the case of irregular (application specific) topology, the parameters are listed in the Table 4.2.

Table 4.2 Parameters and their descriptions for irregular topology

Parameter	Value	Description
Regular/Irregular	1 0	Regular topology. (default is 1) Irregular topology.
Area	R	The area of SoC chip in mm^2. (default is 7)
SIM_NUM	0<N	The total clock cycle that simulator runs. It is better to be chosen less than 10. The default is 1. (default is 1)
FW	0<N	It is related to address of source or destination. The flit width is equal to (2*FW+5) per bit. Increasing it extends the size of registers, and so raises resource cost. (default is 8)
NUM_FLIT	1≤N≤256	The number of flits per packet. (default is 5)
FIFO	1≤N≤256	The number of registers in FIFO. Increasing it extends the size of registers and therefore raises resource cost. (default is 2)
BURST	1≤N	The number of packets generated continuously in the burst time. (default is 1)
PAUSE	N	The pause per clock cycle between two burst messages. (default is 0)
GAP_P	0<R≤1	The proportion of PEROD added to itself and makes new PERIOD. (default is 0)
PRN_OUT	B	Show sending and receiving flits on the console screen. (default is 0)
SCALE_P	0<R≤1	The coefficient to scale PERIOD (0= maximum scale meaning calculating

		PERIOD based on the current rating packets, 1= no scale meaning calculating PERIOD based on the current rating packet divided by minimum rating packet in the system). (default is 1)

Following points have to be considered:

- N= natural number {0, 1, 2 ...}.
- R= real number.
- B= {1 Active, 0 Inactive}.
- Either Torus or Mesh should be activated in a time.
- LOAD parameter is only used in locality or uniform traffic.
- Only one of XY, YX, L_PROBE and ODD_EVEN routings should be activated at a time.
- ODD_EVEN routing works only in the case of Mesh topology.
- Only one of the IRREG_TRAFFIC, UNIFORM, FIXED and LOCALITY traffic generators should be activated at a time.
- LOC_ONE, LOC_TWO, LOC_THREE, LOC_FOUR, LOC_FIVE and LOC_SIX are only used in locality traffic and the probability of each one is equal to its number divided by the total of all numbers. For example, If the numbers (9, 5, 4, 3, 2, 1) represent the values of these parameters respectively, the probability of LOC_FOUR is 3/24= 0.125 or 12.5 %.
- SCALE_P and GAP_P are used for irregular traffic only.
- The above parameters are included in at the top file *main_noc.cpp.* All of the parameters can be changed in the file to have new configurations of NoC except the kind of topology and the two parameters N_COL and N_ROW. Users can change these parameters and apply to the simulator as a new configuration without running the User Interface program.

After typing the values of these parameters, User Interface will show the core graph and the core switch graph of NoC on the console, and then it makes the file, *main_noc.cpp*.

4.1.3 Implementation of NoC Simulation program

The NoC Simulator program consists of several modules such as *arbiter, crossbar, demux and FIFO* that construct the hardware, power and area modeling of NoC. These modules are coded on the file *main_noc.cpp*. This file and *core.txt* construct the NoC Simulation program. The execution of NoC Simulator generates a trace file (graph.vcd). Users can use tools to view the waveforms from the trace file and determine whether or not the simulation has succeeded. The program also displays the power, area and performance results on the console screen as given below.

In the case of uniform and locality traffic patterns, following results can be generated.

i. Total number of packets dropped
ii. Flit injection rate
iii. Throughput
iv. Average latency of a packet
v. Offered load
vi. Link utilization
vii. Dynamic architectural power dissipation of routers
viii. Static architectural power dissipation of routers
ix. Total areas of routers and links
x. Dynamic transactional power dissipation of routers
xi. Static transactional power dissipation of routers
xii. Architectural power dissipation of links
xiii. Transactional power dissipation of links

In the case of irregular traffic, following simulation results can be obtained.

i. Total number of outgoing flits
ii. Total number of incoming flits

iii. Throughput
iv. Minimum PERIOD
v. Total latency of a packet
vi. Throughput per clock cycle
vii. Average latency of a packet
viii. Dynamic architectural power dissipation of routers
ix. Static architectural power dissipation of routers
x. Total area of routers
xi. Dynamic transactional power dissipation of routers
xii. Static transactional power dissipation of routers
xiii. Architectural power dissipation of links
xiv. Transactional power dissipation of links
xv. Total area of links

4.2 Contention Due to Multi Points

Before starting the simulation experiment, we first discuss about the source of contention in NoC. This contention is created due to spatial distribution of the source and sink cores and we call it point-to-multipoint (PTMP) contention. The PTMP contention is created when the instant rate of packet production is more than the instant rate of packet acceptance. It may happen when a source sends packets to multiple sinks or a sink receives packets from multiple sources. In other words, if each source sends packet to only one sink and each sink receives packet from only one source, NoC will not have PTMP contention. There are large numbers of solutions to solve the contention in NoC such as by adding virtual channel, using routing mechanism, changing buffer size, etc. However, these solutions do not solve the PTMP contention problem. For example, changing the configuration of NoC structure might only alleviate the PTMP contention but will not eliminate it completely even when we add more virtual channels. The

only solution to this problem is the temporal changing in the traffic where the sources can pause the traffic to let the waiting packets reach their destinations.

4.3 Configuration for the First Set of Simulation Experiment

As mentioned in section 3.8, the user should first initialize the NoC and its traffic pattern. The following configuration is an example based on some typical resources to conduct an experimental simulation.

We use a 4×4 regular topology (Mesh or Torus) and application specific topology (related to MPEG4 and AV benchmark applications).

We use routing mechanisms of XY with one channel, YX with one channel, XY with four virtual channels, LP with one channel, and Odd-Even with one channel. For simplicity we name them XY, YX, VC4, LP, Odd-Even respectively.

For traffic generation, we employ three traffic patterns such as uniform, locality, and application-oriented. In application-oriented traffic, the simulation time is chosen a particular value of time which is parameterized to PERIOD. PERIOD is the minimum guaranteed time in which every source sends at least one packet to its destination (for more detail see section 4.7.1). In uniform and locality traffic, the simulation time is chosen to be 100 (or any other suitable value), and the traffic generator dictates synthetic traffic to the sources. In other words, the sources inject packets into the network with a constant rate. In uniform and locality traffic, each node sends packets to only one destination node at a time, and on the contrary, each node receives packet from only one destination node at a time i.e. no PTMP contention in the NoC. In the case of uniform traffic, a network node sends packets to other nodes with the same probability. In the case of locality traffic, a network node sends packets to its neighbouring nodes with a higher

probability. The number of VCs per physical channel is maximum four and the depth of a VC is selected as two. One message contains only one packet and each packet is split into five flits. The network is asynchronous, and there is no relation between the clocks of source, sink or router modules.

4.4 NoC Simulation for Uniform Traffic

In this section, different routing mechanisms are evaluated and the advantage of adaptive routing mechanism such as LP routing is demonstrated by using uniform traffic pattern. As mentioned earlier, the uniform traffic is a packet flow pattern in which a source node sends packets to other nodes with the same probability [33]. The source in this pattern is a greedy source as it produces packets at the highest rate possible and at the earliest opportunity possible [32]. Figure 4.2 shows four graphs of performance metrics (i.e., packet drop, throughput, average latency, and link utilization) vs. flit injection rate. These graphs demonstrate three routing mechanisms of XY, LP and VC4 by using uniform traffic in a 4×4 torus topology. For better understanding of these graphs, the following points determine which values in each graph are optimums. In the packets dropped graph, the optimum value is zero i.e. no packet should drop in the NoC. For throughput graph, the optimum throughput happens when it is equal to the injection rate. For latency graph, the optimum value happens when latency is equal to five clock cycles that is equal to the size of a packet.

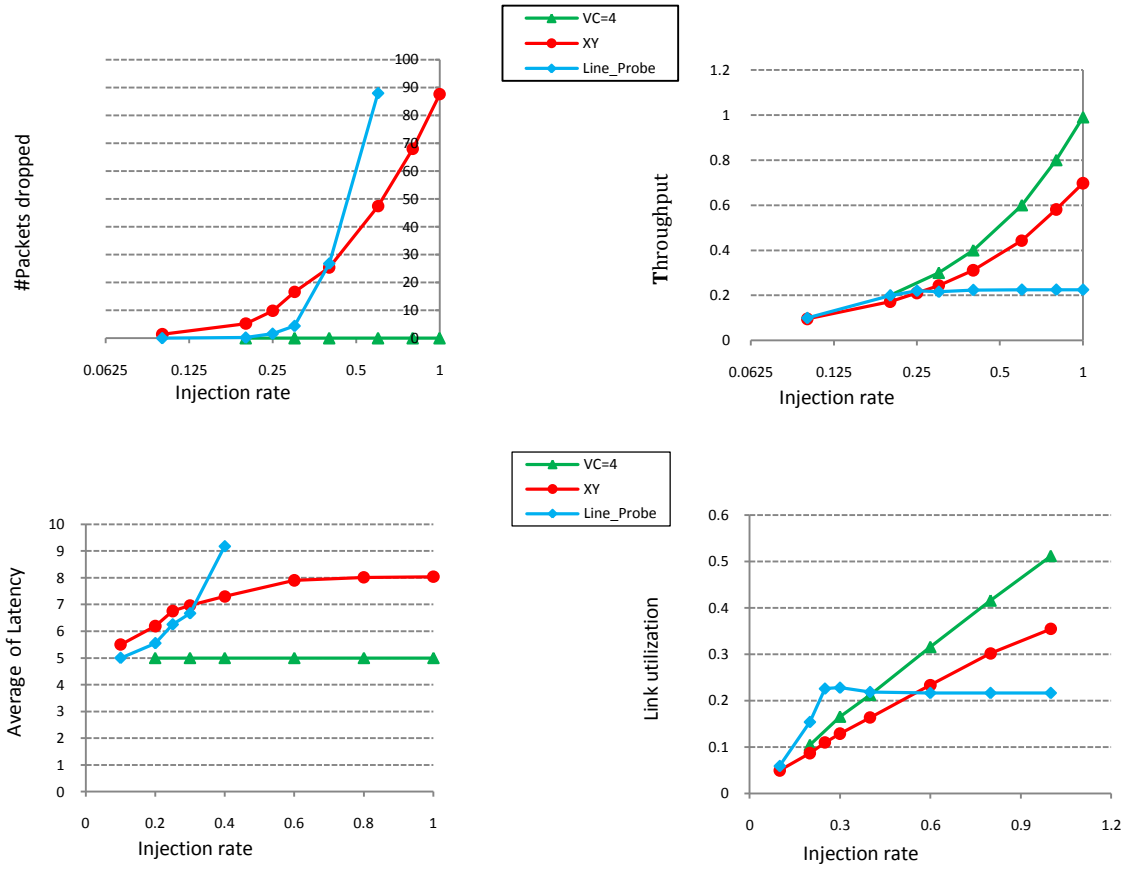


Figure 4.2: Graphs in Uniform traffic, Torus topology

In the case of link utilization graph, lower link utilization indicates the more effective system, however, the throughput should also be considered.

As it can be easily envisaged in the figures of 4.2, VC4 is contention free that means the traffic does not have latency, packet drop is zero, and throughput is the highest. This claim can be verified by two methods. In the first method, each node sends packets to only another node at a time so that the instant rate of incoming packets are equal to the instant rate of outgoing packets (i.e., there is no PTMP contention). Secondly, in small NoCs (e.g. 4×4), we have figured out that no more than four messages pass through a link at a time. Therefore, four virtual channels are enough to handle all the messages without contention. This capability of VC4 helps

us to consider it as a base to compare two other routing mechanisms. Therefore, between two graphs for LP (line probe) and XY, the graph that is closer to VC4 has better performance. When we consider the three graphs related to throughput, packet drop and latency of Fig. 4.2, it can be concluded that LP routing has advantage over XY up to 25% of the injection rate. However, the results are opposite for link utilization. It is logical conclusion that to acquire a better performance, the system should spend more resources or sacrifice for one performance factor in place of the other. The LP routing improves the delay and throughput of system at the expense of increasing link utilization. It can also indirectly affect the other two important metrics such as delay and power consumption. Finally, we can conclude that the network shows significant performance improvement for LP routing as compared to XY at 25% of cases in a uniform traffic.

4.5 Locality Traffic Base Simulation

The on-chip network shows significant performance improvement if the traffic is more locally distributed [33]. Moreover, locality traffic is more realistic than uniform traffic because a designer tries to put the source and destination at the minimum distance. The best NoC design will be where each node sends packet to near neighbour as shown in Figure 4.3. These two figures depict throughput and delay related to the injection rate for three type of traffic: locality, non-locality and uniform. Figure 4.3 indicates the advantage of locality traffic over the two other traffics.

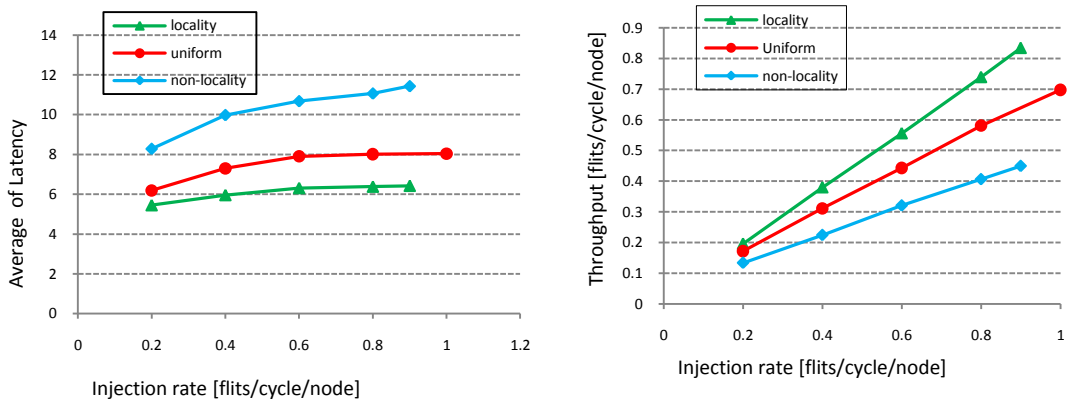


Figure 4.3: Graphs in uniform and locality traffic

4.5.1 Contention Free in LP Routing

In the case of a 4×4 torus topology, the maximum distance between two nodes is four. Thus, there can be four coefficient probability in the NoC. For maximum locality in the system (i.e. each source sends packet to its neighbour), we expect a contention free system. Maximum locality is initialized as LOC_ONE=16, LOC_TWO=0, LOC_THREE=0, LOC_FOUR=0 (see section 2.7.1) and shown as 16_0_0_0 in Table 4.3. We start decreasing the locality and at each step execute the simulator and measure the packet drop. If a packet is dropped, it means that there has been a contention in the NoC. Table 4.3 demonstrates these results. For obtaining these results, we first decrease the probability of first neighbour and increase the probability of second neighbour. We reach border 8_8_0_0 without contention. The setup coefficient 8_8_0_0 indicates the probability of sending packet to the first and second neighbours as eight and probability of sending packet to third and forth neighbours is zero. We try to increase the probability of third and forth neighbours where we reach the results given in Table 4.3.

Table 4.3 Border of contention free in LP routing with Specification: Sum_num=100, Torus topology 4×4, Virtual channel=1, FIFO = 2, locality traffic, NUM_FLIT 5, XY and LOAD=0.9						
Coef	Packet drop	Flit injection rate	Throughput	Average latency of a packet	Offered load	Link utilization
16_0_0_0	0	0.9	0.9	5	0.25	0.25
15_1_0_0	0	0.9	0.9	5	0.28125	0.28125
10_6_0_0	0	0.9	0.9	5	0.375	0.375
8_8_0_0	0	0.9	0.9	5	0.4	0.4
8_7_1_0	0	0.9	0.9	5	0.426562	0.426562
12_2_2_0	1	0.9	0.896875	5.09059	0.397969	0.397969
12_2_1_1	0	0.9	0.9	5	0.332812	0.332812

The following simulation results show 33% of the cases without contention in LP routing.

Total number of cases that each node sends packet to its first neighbours = 48

Total number of cases that each node sends packet to its second neighbours = 68

Total number of cases that each node sends packet to its third neighbours = 48

Total number of cases that each node sends packet to its forth neighbours = 16

The total number of cases= 48+68+48+16= 180

Table 4.3 coef 16_0_0_0, the number of cases without contention = 48

Table 4.3 coef 8_8_0_0, the number of cases without contention= 9

Table 4.3 coef 8_7_1_0, the number of cases without contention= 2

Table 4.3 coef 12_2_1_1, the number of cases without contention= 1

The total number without contention= 48+9+2+1= 60

The percent of contention free ability= 60÷180 × 100= 33%

Table 4.4 has the same *coef* like table 4.3 but for XY routing. Comparing these two tables shows the significant improvement of LP routing.

Table 4.4 XY routing in locality traffic with Specification: Sum_num=100, Torus topology 4×4, Virtual channel=1, FIFO = 2, locality traffic, NUM_FLIT 5, XY and LOAD=0.9

coef	Packet drop	Flit injection rate	Throughput	Average latency of a packet	Offered load	Link utilization
16_0_0_0	6.2	0.9	0.8775	5.87296	0.239844	0.238704
15_1_0_0	14.6	0.9	0.851875	6.21214	0.253594	0.252706
10_6_0_0	39	0.9	0.774375	6.87149	0.287969	0.286715
8_8_0_0	32.2	0.9	0.794375	6.54496	0.32125	0.319611
8_7_1_0	43.6	0.9	0.75875	6.88462	0.314375	0.312745
12_2_2_0	28.6	0.9	0.808125	6.52506	0.309375	0.308545
12_2_1_1	21	0.9	0.834375	6.42397	0.300469	0.300469

4.5.2 Advantages of Locality Traffic

Figure 4.4 shows the graphs of performance metrics for three routing mechanisms XY, VC4, and LP using locality traffic with coefficient 8-7-1-0. For packet drop, throughput and latency figures of 4.4, it can be concluded that the LP and VC4 graphs are completely overlapped. We already mentioned that VC4 routing is contention free and LP routing is also contention free when it only uses one FIFO buffer. However, the LP routing utilizes more links. In previous sections, it was discussed that contention among packets, two or more packets may need to pass along the same link. The XY routing only let one packet pass and others should stay in the queue.

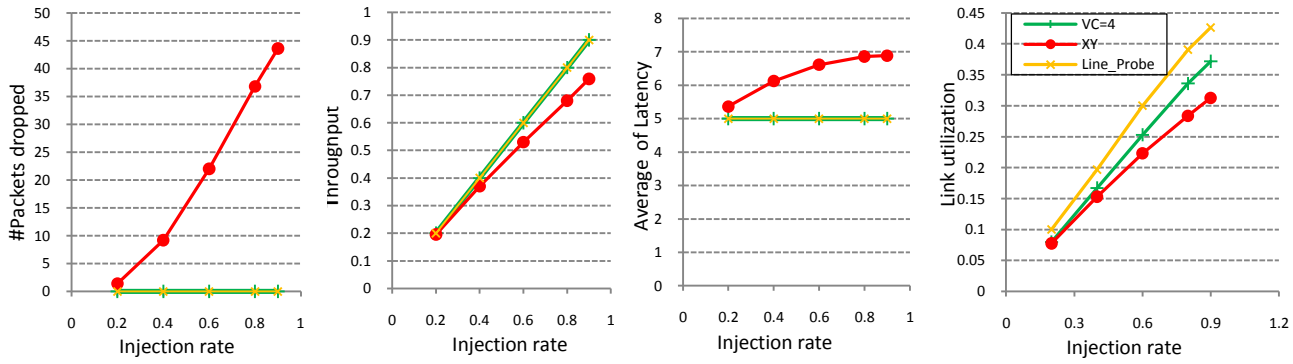


Figure 4.4: Graphs in locality traffic, Torus topology, coef 8-7-1-0

The VC4 routing allows four packets pass simultaneously while LP routing allows one of the packets pass via the link and rest are sent via other available links that may not yield shortest paths. This causes the link utilization is increased in the system. In other words, LP routing takes advantage of extra resources in torus topology and improves the performance of NoC.

4.6 Mesh Topology Simulation

So far we have not discussed Odd-Even routing because its original mechanism works for a mesh topology. We implemented it in a torus topology, but it encountered the looping process in some cases. Wang Zhang et al. concluded that Odd-Even routing mechanism fits well with 2D 3×3 mesh topology even better than XY routing mechanism for constant bit rate traffic condition [20]. Our results of Fig. 4.5 that consists of the throughput and latency graphs confirm their conclusions. Figure 4.5 shows output results of XY, VC4, LP, and Odd-Even routing in mesh topology by using locality traffic with coef 16-0-0-0-0-0. In this experiment, we found that VC4 and Odd-Even routings are faster than XY and LP routing. LP routing is slower than others because the dimension of network is small (4×4) to efficiently implement the LP routing. In other words, the four nodes in the corners do not have any cooperation in the LP routing, and eight side nodes have semi-cooperation in the LP routing. Only four nodes out of sixteen nodes are fully involved in LP routing that increases link utilization. LP routing performs better when link utilization is low, or in other words when the network has more free resources.

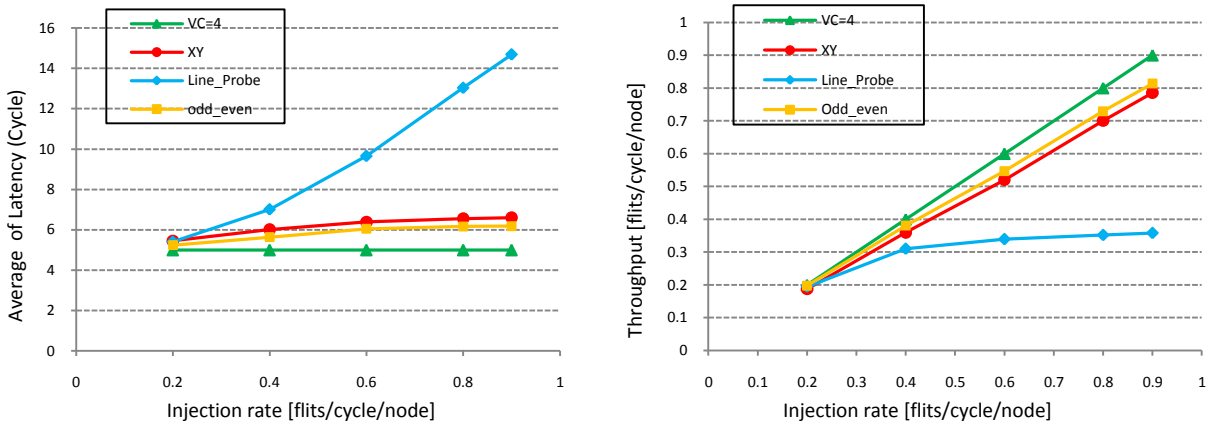
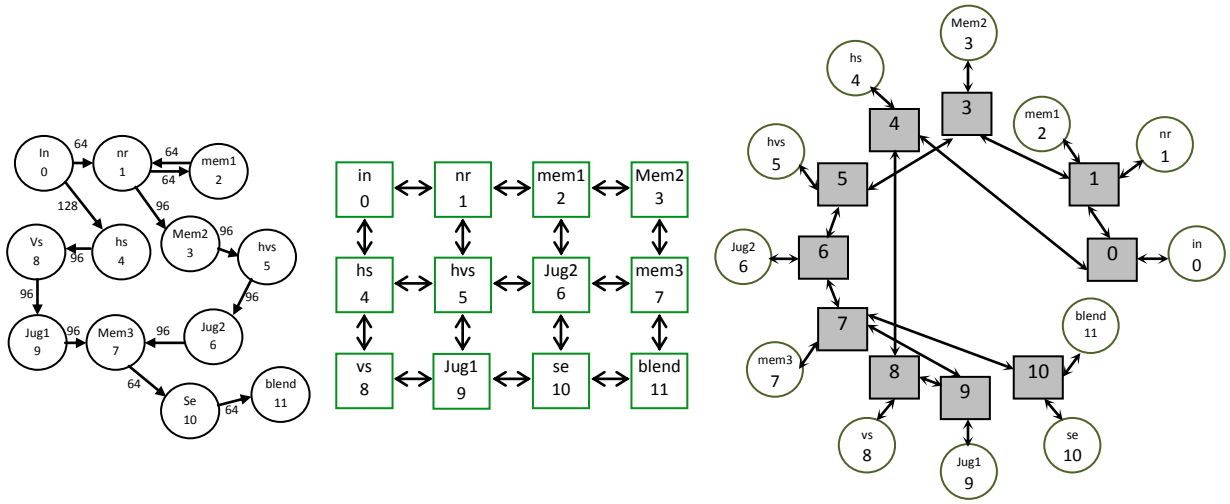


Figure 4.5: Graphs in locality traffic, Mesh topology, coef 16-0-0-0

4.7 Simulation for Application-Oriented Traffic

The ultimate goal of an NoC designer is to design the NoC system based on application. The application can be mapped on regular or irregular topologies. For regular topology, the cores of an application are numbered based on either a specific methodology or randomly. Then the application cores are mapped on the nodes of a 2D topology. Figures 4.6.a and 4.6.b demonstrate this methodology for an application. The traffic generator should generate the packet flow based on the application (as it is called application-oriented traffic). The only consideration is that the number of nodes in 2D topology should be more than the number of cores of the application. The simulator automatically constructs the 2D topology of application and generates packet flow in accordance with the *core.txt* file of the application. For irregular topology, if we put a switch at every node of the core graph of the application, we have mapped the application on the application specific topology that has an irregular pattern. Figure 4.6 depicts this methodology for an application. The router in an irregular topology is modeled to have dedicated output for each input channel. The simulator automatically builds the application specific topology and generating packet flow in accordance with the application *core.txt* file. Some SoC applications

have been used in the past and due to their repeated usages, they have become benchmark applications for current and future research. In the next section, we investigate two applications such as MPEG-4 decoder [49] and AV Benchmark applications [14]. The MPEG-4 application can be mapped to a 3×4 2D topology and an AV application to a 4×4 2D topology. In section 4.11, we experiment the SoC applications as followings: a MPEG4 decoder (MPEG4), a Multi-Window Display (MWD), a Video Object Plane Decoder (VOPD) and an enhanced VOPD application, called DVOPD with the capability to decode two streams in parallel.



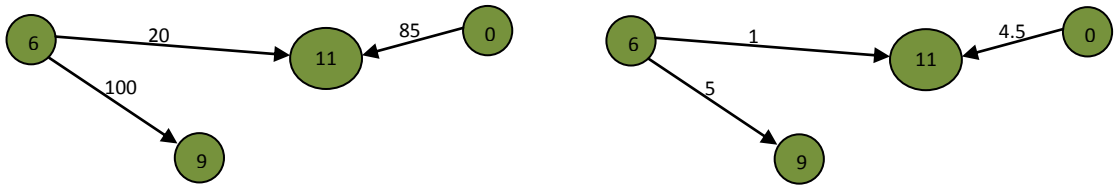
(a) Core graph (b) Mesh mapping (c) Irregular mapping

Figure 4.6: Application mapping on regular or irregular topologies

4.7.1 Simulation Time for Irregular Traffic

To determine optimal condition for which the system can reach maximum throughput (100%) for an irregular traffic, the simulation time of simulator should be initialized properly as following. Consider Fig. 4.7.a showing a branch of nodes for an application implemented in the simulator, and the simulation time is selected equal to the maximum rate of packets in the branch i.e. one hundred. Moreover, assume that each packet is sent at one clock cycle, and each source

sends all of its rating packets once during simulation time. During simulation time, when core 6 sends packets to core 9, it does not have time to send packet to core 11. When core 11 finishes receiving packet from core 0, it only can receive fifteen packets from core 6. To prevent such a problem, we define a term PERIOD which is the minimum guaranteed time for which every source sends at least a packet to its destination. To calculate this time, we should scale the flow to one packet for minimum rate. Figure 4.7.b depicts this case.



(a) Source of contention (b) Scaling flow to minimum rate

Figure 4.7: Source of contention and scaling flow

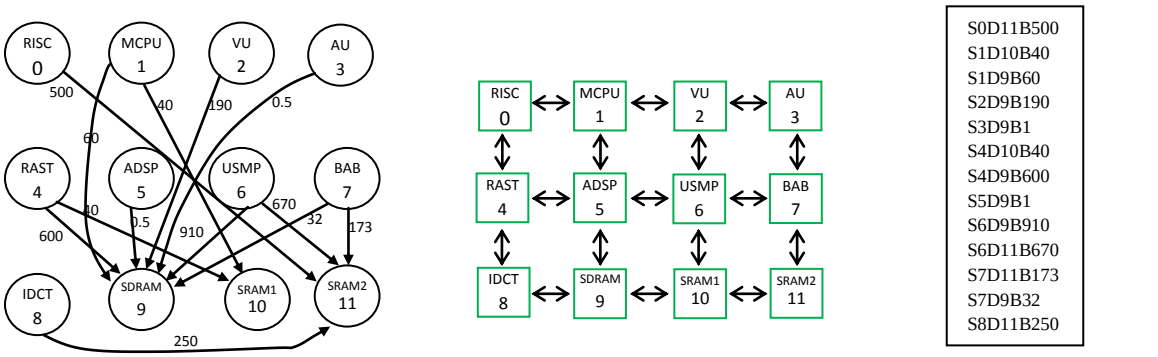
In Fig. 4.7b, core 6 needs six clock cycles to send all its packets, and core 11 needs 5.5 clock cycles to receive all its packets. As we do not have decimal point in clock cycle, the number 5.5 is rounded to six. Therefore, to guarantee at least one packet for every source, PERIOD is chosen equal to six. If we multiply PERIOD to minimum rate of twenty we get 120 clock cycles. This amount of time is the minimum guaranteed time during which every source sends the total of its dedicated packets to its destinations once. When the traffic is irregular, FAANOS automatically calculates PERIOD and stores it as the minimum simulation time. The parameter, SCALE_P lets users choose the simulation time equal to PERIOD when SCALE_P=1, or the simulation time equal to multiplication of minimum rate and PERIOD when SCALE_P=0. In the above example, when SCALE_P=0, then the simulation time is equal to 120 and when SCALE_P=1, then the simulation time is equal to six.

4.7.2 NoC Simulation for MPEG-4 Decoder

Figure 4.8.a shows the core graph of MPEG-4 decoder with communication (in MB/s). The mapping of MPEG-4 on the mesh topology is given in Fig. 4.8.b. The context of *core.txt* file is given in Fig. 4.8.c which is based on the methodology presented in section 4.1.1. In this application, each source sends packet with different rate and the spatial distribution of traffic is fixed and based on the MPEG-4 decoder application. Figure 4.9 shows the results of throughput for five routing mechanisms, XY, YX, LP with 1 channel (LP), XY with 4 virtual channels (VC4), and ODD-EVEN with 1 channel (OE) in different configurations as following.

- "Torus" means the default configuration of a regular topology with ROW=3, COLUMN=4, SIM_NUM=1, IRREG_TRAFFIC =1 and UNIFORM=0.
- "Torus, SIM=2" means "Torus" with SIM_NUM=2.
- "Torus, r_robin" means "Torus" with R_ROBIN =1.
- "Mesh" means the same configuration as "Torus" with MESH=1 and TORUS=0.
- "Torus, 4×3" means the same configuration as "Torus" with ROW=4 and COLUMN=3.

The graphs of Fig. 4.9 show that LP routing has the worst performance among all these routing mechanisms. As mentioned earlier, this is due to the fact that LP routing is not suitable for NoC systems with PTMP contention. Figure 4.8.a easily shows this problem for MPEG-4 where seven sources send packets to sink 9 concurrently. Sink 9 can receive the packets from one source and the remaining packets travel in the network that will increase the link utilization leading to lock a network (for example in "torus, r_robin" configuration). The following section verifies our claim. We will verify this claim in the next paragraph.



(a) MPEG4 core graph (b) Mesh mapping (c) MPEG4 txt core table

Figure 4.8: MPEG4 core graph and its mapping to 2D topology

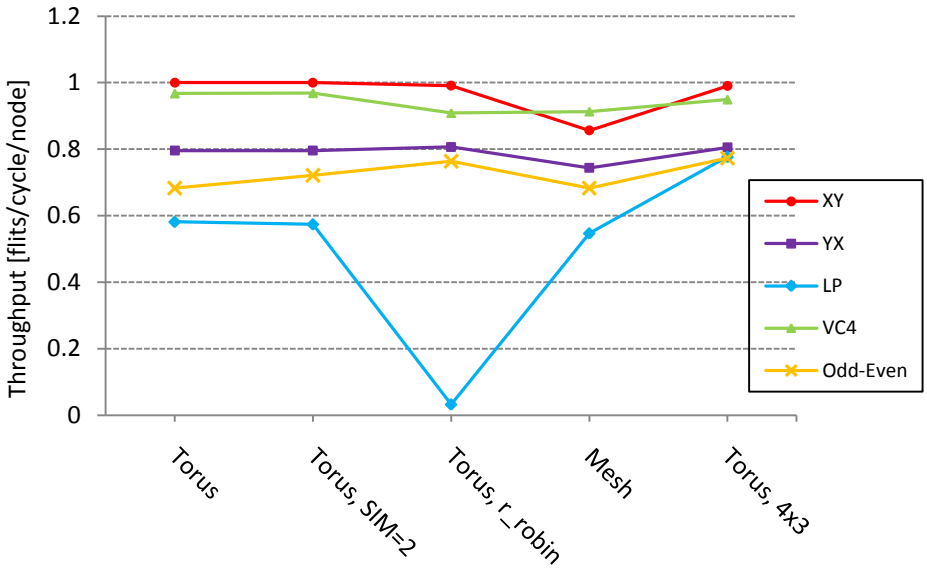


Figure 4.9: Throughput results of MPEG-4 Decoder application

To verify the claim, we invert the direction of packet flow in the core graph of MPEG-4. Figure 4.10 shows the reversed core graph and the throughput results of MPEG-4 that verifies our claim. The LP routing is faster and has second place after VC4. In this system as there is no PTMP contention, the instant rate of incoming packets is less than the instant rate of outgoing packets. It provides the best condition for LP routing mechanism.

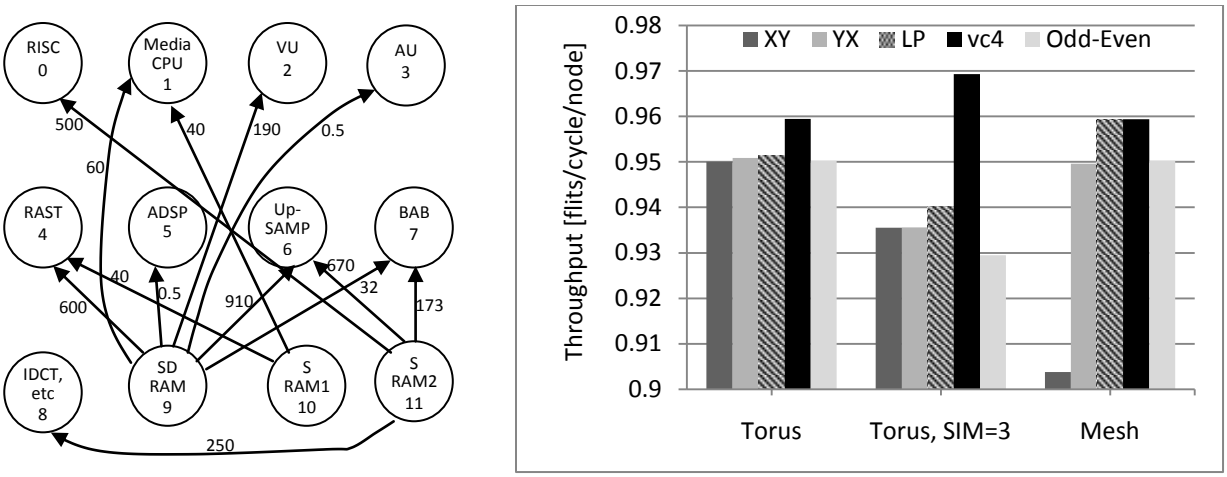


Figure 4.10: Reversed core graph and its throughput results of MPEG-4 decoder

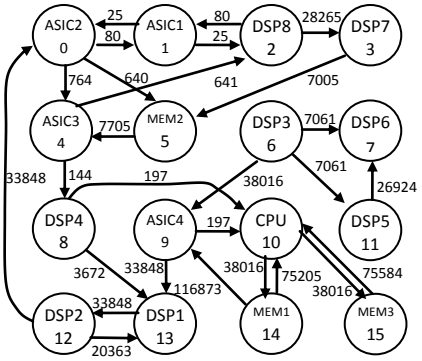
4.7.3 NoC Simulation for AV Benchmark

AV Benchmark application core graph is given in Figure 4.11.a where Fig. 4.11.b shows the mapping of this application on to a 4×4 torus topology. Figure 4.11.c is the context of core.txt file which is created as an input to the simulator. In this NoC system, each source sends packet with different rate, and the spatial distribution of traffic is based on the AV Benchmark application. Figure 4.12 depicts the simulation results of throughput for various routing mechanisms of XY, LP, VC4 and ODD-EVEN (one channel) in different configurations as following.

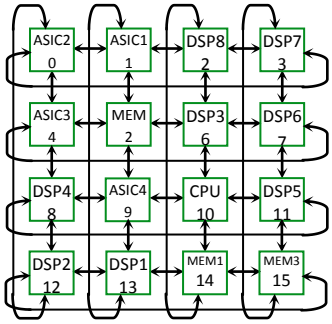
- "Torus" means the default configuration (see section 4.1.2) of a regular topology with SIM_NUM=1, IRREG_TRAFFIC =1 and UNIFORM=0.
- "Torus, reverse" means the same configuration as "Torus" for reverse core graph (inverting the direction of packet flow in the core graph of application).
- "Mesh" means the same configuration as "Torus" except on torus topology.

- "Mesh, reverse" means the same configuration as "Mesh" for reverse core graph.

The bar graphs of Fig. 4.12 show that the LP routing has the worst performance among these four routing mechanisms. Again the same issue as in MPEG-4 application has happened. LP routing is not suitable where there is PTMP contention. Even by inversing packet flow in the core graph of application again the LP routing has the worst performance in the system. This is because in reversing core graph again there is PTMP contention.



(a) Core graph of AV Benchmark



(b) Torus mapping

S0D1B80	S8D10B197
S0D4B764	S8D13B3672
S0D5B640	S9D10B197
S1D0B25	S9D13B33848
S1D2B25	S10D14B38016
S2D1B80	S10D15B38016
S2D3B28265	S11D7B26924
S3D5B7005	S12D0B33848
S4D2B641	S12D13B20363
S4D8B144	S13D12B33848
S5D4B7705	S14D9B116873
S6D7B7061	S14D10B75205
S6D9B38016	S15D10B75584
S6D11B7061	

(c) Core text of AV

Figure 4.11: The core graph, torus mapping and core txt of AV Benchmark application

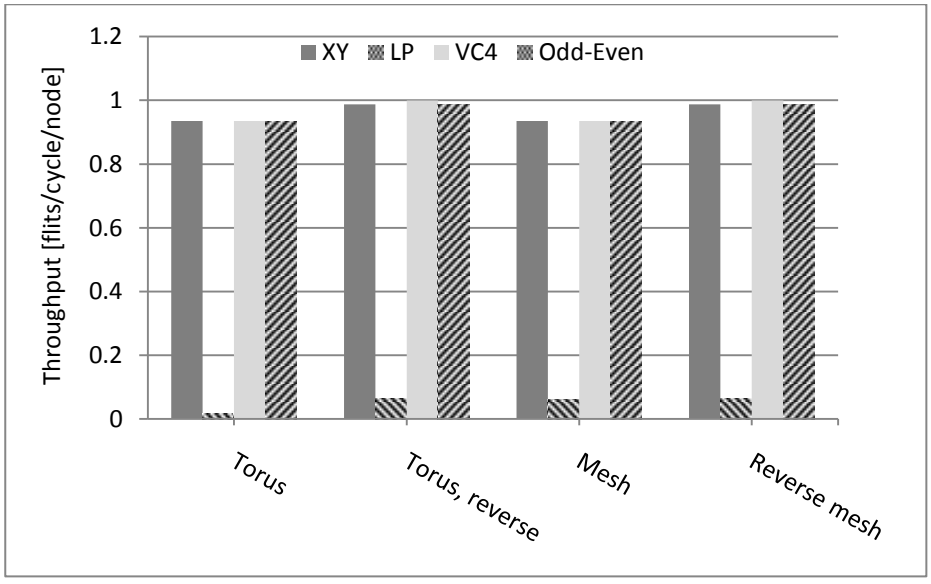


Figure 4.12: Throughput results of AV Benchmark

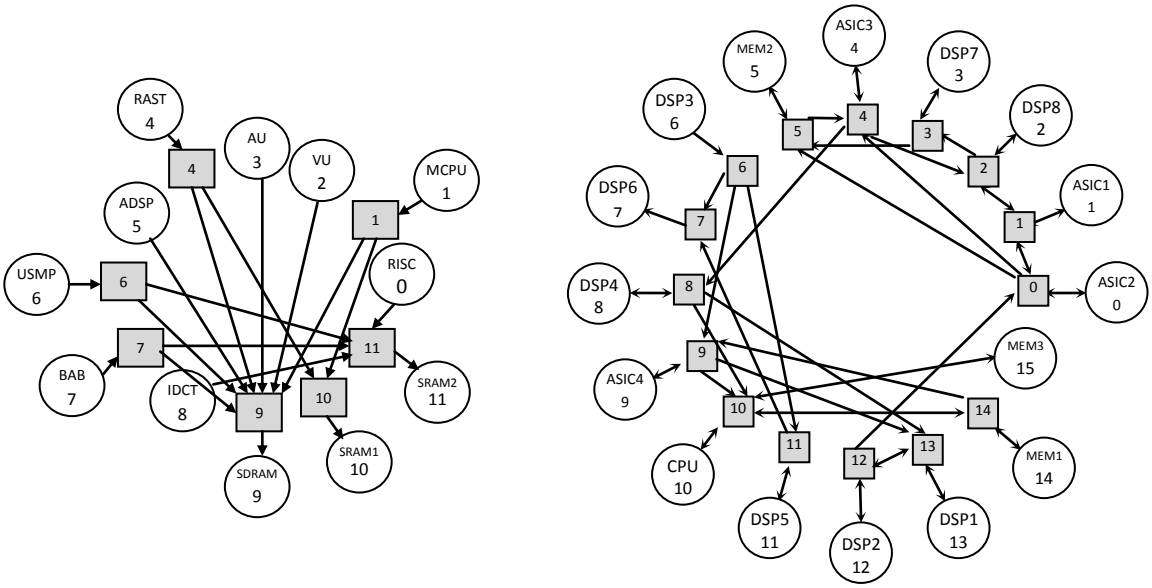
Figure 4.11a indicates this problem. Core 0 sends packet to three other cores at the same time and it also receives packets from two other cores at the same time. Core 0 can receive the packets from one source and the remaining packets will wonder around in the NoC that will increase link utilization.

4.7.4 NoC Simulation for Irregular Topologies

Figures 4.13a and 4.13b show the mapping of MPEG-4 and AV application to regular topology. Figure 4.14 is the throughput results of MPEG-4 and AV application in regular or irregular topology. The configuration of the simulator is selected as follows.

- "Torus" means the default configuration of Table 4.1 (see section 4.1.2).
- "Mesh" means the default configuration of Table 4.1 with Mesh=1 and Torus=0.
- "Irregular" means the default configuration of an irregular topology (see Table 4.2).

The advantage of regular topology over the irregular topology for MPEG-4 application and the equal is shown in Fig. 4.14. However, in the case of AV application, there is no improvement in throughput as depicted in Fig. 4.14. This is obvious as in the regular topology more resources such as links and channels are employed. These extra sharing resources improve the performance of network system. As in MPEG-4 application, the implementation in Torus topology has the largest throughput.



(a) Mapping of MPEG-4 on regular topology

(b) Mapping of AV on irregular topology

Figure 4.13: Mapping of the MPEG-4 and AV application on irregular topology

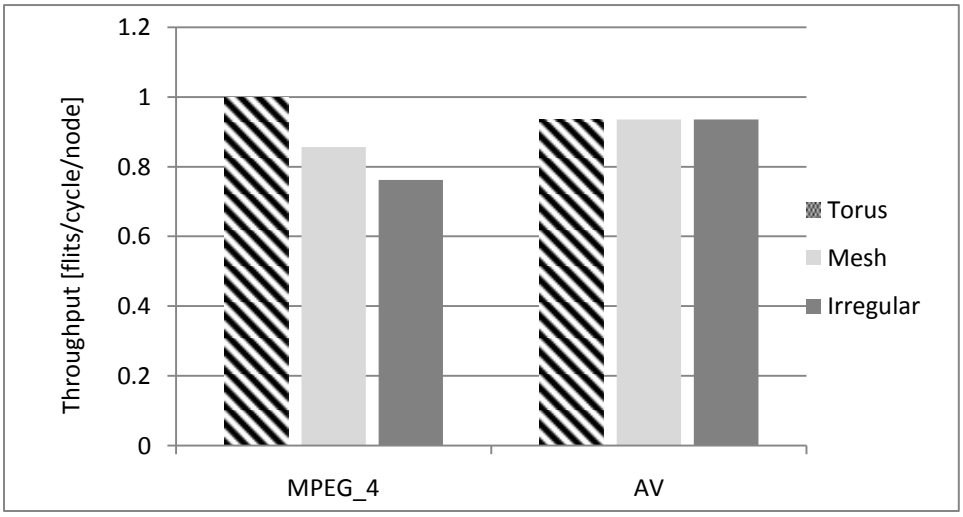


Figure 4.14: Throughput results of MPEG-4 and AV application in regular and irregular topology

4.8 Power and Area Chip

In this section, we present the second set of experiment by also considering the power consumption and chip area of NoC. We are not seeking any claim to prove only the behaviour of some characteristics of NoCs will be investigated. These analytical investigations demonstrate not only the capability of our simulator but also the challenges in response to satisfaction of NoC constrains. We present the investigation on 2D topology in section 4.8.1 and on application specific topology in section 4.8.2.

4.8.1 Simulation for 2D topologies

We have selected the test cases based on several criteria such as the number of routers ranging from 9 to 36 and the number of virtual channels (one and four). The goal is to study the impact of NoC size and number of virtual channels on the power and area of NoC for torus topologies. Specifically, we are interested in the architectural and transactional power and area breakdown. We assumed a 90nm technology and a clock frequency of 1GHz. The area occupied by cores is fixed and assumed to be 9 mm^2. The other specifications of NoC are as follows.

- Number of simulation=1000 cycle.
- Depth of FIFO = 2.
- Traffic pattern = Uniform.
- Size of each packet = 5 flits.
- Routing strategy = XY mechanism.
- Size of each flit =31 bits.

The simulation results of synthesizing different torus NoCs are reported in Fig. 4.15. Each histogram is divided into two zones: one virtual channel (VC=1) and four virtual channels (VC=4). Each zone contains four bars and each bar for different NoC sizes such as 3×3, 4×4, 5×5 and 6×6. We choose different NoCs to show the effect of NoC sizes on the NoC metrics. Since the NoC size determines the number of routers in 2D topology, NoC with size 3×3 has 9 routers and so on for other mentioned NoCs. Therefore, we expect a variation with slope of 9, 16, 25, 36 (or 1, 1.8, 2.8, 4) between the metric results of these NoCs. In Fig. 4.15b, it can be easily grasped that the router area is a function of NoC size in both VC=1 and VC=4 zones. It is logical because routers have fixed size (5×5) in torus topology, and the NoC size has a linear relation with the number of routers. Therefore, the slope of increasing in the router area is like the NoC size in both zones of Fig. 4.15b. However, the slope of increasing in the link area is not the same as router area. This is because the area of link is a function of the total link length that is estimated based on SoC area, and the SoC area consists of the area occupied by the cores and the routers; the area occupied by the routers is a function of the NoC size, but the area occupied by cores is assumed to be fixed and 9 mm^2; this fixed number causes the slope of increasing in link area not to be the same as router area in both VC=1 and VC=4 zones. We also see this issue in the architectural power graph. In Fig. 4.15c, the router architectural power is a function of NoC size in both VC=1 and VC=4 zones. It is logical because the router size (5×5) is fixed, and the NoC size has a linear relation with the number of routers. Therefore, the slope of increasing in the router architectural power is like the NoC size in both zones. However, the slope of increasing in link architectural power is not the same as router area. As mentioned before, this is because the cores area is fixed and is not a function of the NoC size; this fixed number causes the slope of increasing in link architectural power not to be the same as NoC size in both VC=1 and

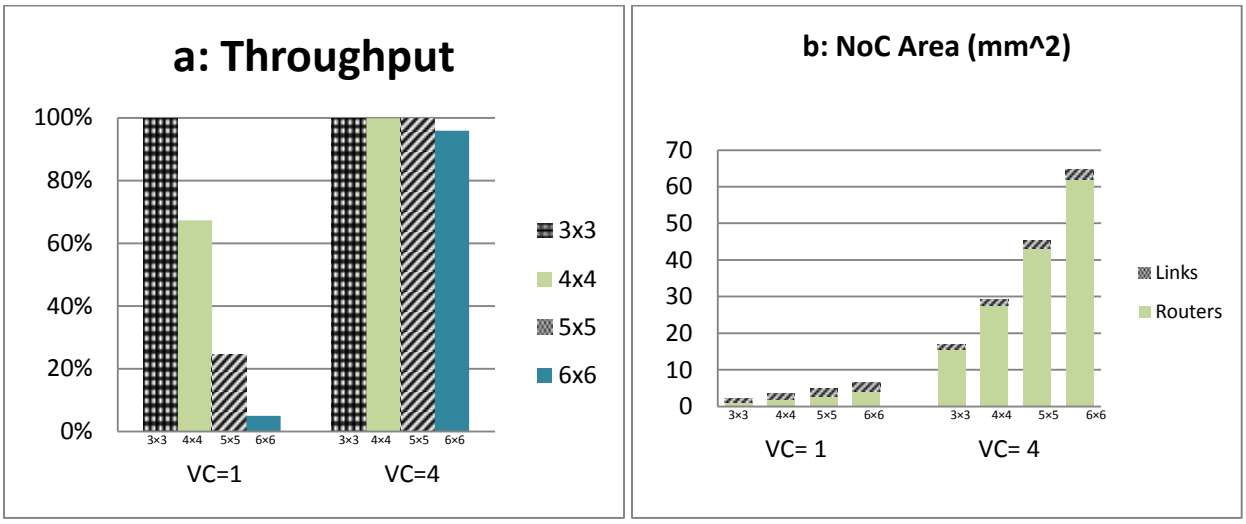
VC=4 zones. In transactional power graph, the behaviour of bars are different in the VC=1 and VC=4 zones. In the VC=4 zone of Fig.4.15d, which the routers have 4 virtual channels, there is not contention in the NoCs and the throughputs are almost 100% as depicted in Fig. 4.15a. The transactional power is a function of the rate of router and link transactions in NoC. The rate of router transaction when throughput is 100% is a function of NoC size. In other word, when NoC size increases, the number of cores will increase and more transactions occur in NoC, so when throughput is 100%, the router transactional power will be a function of the NoC size. However, the same issue as mentioned before is happen for link transactional power. The constant value of core size changes the corresponding relation between the link transactional power and the NoC size. In this case, the slope of increasing in transactional power bars resembles the architectural power in VC=4 zone of Fig. 4.15c and Fig. 4.15d.

In the VC=1 zone of Fig. 4.15d, there is no relation between transactional power and NoC size. We can find the reason of this difference in throughput graph as depicted in Fig. 4.15a where the average of throughput is less than100%. In this case, we cannot estimate the behaviour of NoC transactional power. In other words, in a deterministic (XY) routing strategy, the higher throughput represents the higher rate of transaction that leads to the larger amount of transactional power occurs in NoC. For example, the bar 6×6 is less than other bars because the corresponding bar in Fig. 4.15a is the least.

Two other points that are highlighted in Fig. 4.15 are the inconsiderable static power and considerable link power and area. The static power in architectural power is very small that we can ignore it in our design. However, in transactional power especially in the VC=1 zone, it can be a considerable amount in compared to dynamic power. Therefore, it needs to be available in NoC design. The link characteristic is important because sometimes it includes a significant

amount of NoC power. For example, in Fig. 4.15c, the link architectural power includes over one-third of each bar.

By considering the above mentioned points, we can conclude that the architectural power consumption and the area occupied by the network are the increasing functions of the structural specification of NoC such as size and number of virtual channel in 2D topologies. However, The NoC transactional power is the increasing functions of the structural and transactional specifications of NoC such as size, number of virtual channels, and throughput. The transactional power is close to realistic power of NoC that can be helpful to determine the bottle neck of NoC power, and the architectural power determines the maximum power that is consumed in the NoC. This will be very helpful for the earliest step of pre-design of an NoC. For example, in the VC=4 zone which represents a guaranteed contention free NoC, the average area occupied by the routers is 9.1 times more than the average area in the VC=1 zone. However, the variation in architectural power are less i.e. the average power in the VC=4 zone is 1.7 times more than the average power in the VC=1 zone. This gives an idea to designer which increasing virtual channel from one to four imposes more cost on area than power.



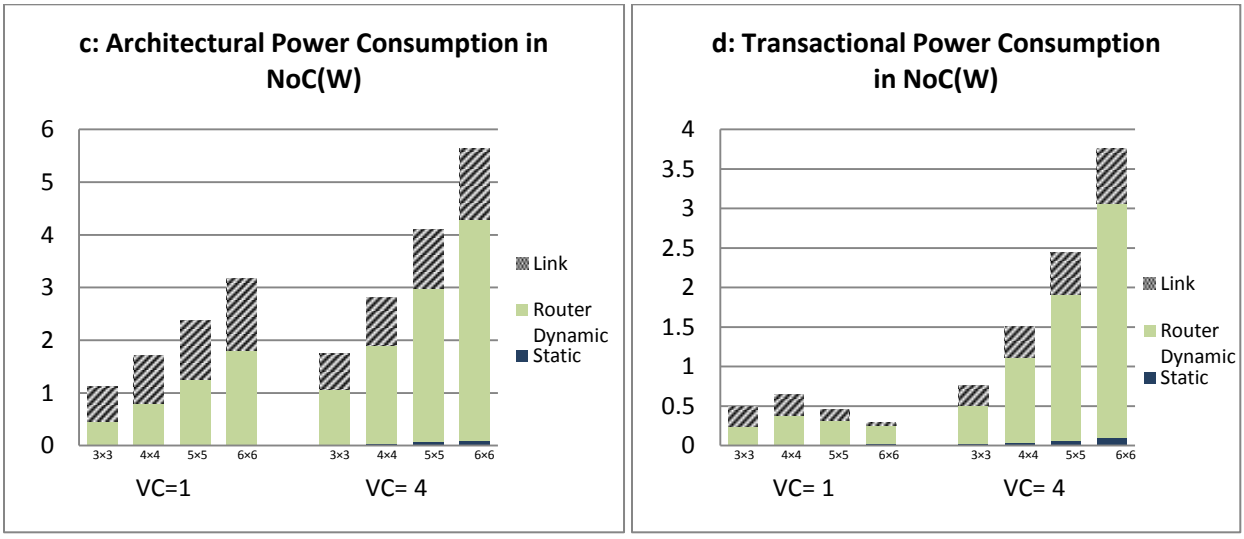


Figure 4.15: NoC synthesized results for the 3×3, 4×4, 5×5 and 6×6 Torus topologies. Power is expressed in Watts, area in mm^2 and throughput in percent. We used the following notation: VC for virtual channel, 3×3 for 3×3 torus topology,... and 6×6 for 6×6 torus topology.

4.8.2 Simulation for NoC Applications

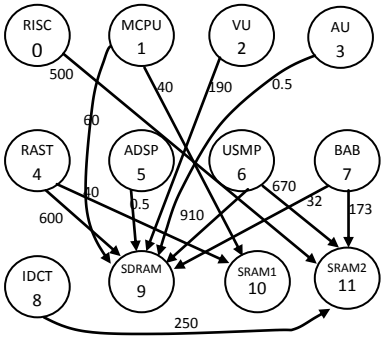
Table 4.5 lists the SoC applications that we have synthesized in our simulator. The second column of this table presents the chip area that each application occupies. Because the area size of NoC is a characteristic that we measures during our experiment and is a function of other NoC characteristic, we consider the values in the second column of Table 4.5 as the chip area that the cores occupy in each application. The intent of our experiments is to study the impact of features of these applications on the synthesized NoC. We are specifically interested in the architectural and transactional power and area breakdown. We have selected the test cases based on several criteria.

- The number of IP cores, ranging from 12 to 26, and the size of IP cores, as large as 12 mm^2;
- The topologies which the applications are mapped namely Mesh, Torus, and application specific topology [4].

Table 4.5 Characteristics of selected SOC applications		
Name	Area (mm2)	Ref.
MPEG4	3 ×2.35	[50]
MWD	3 × 4	[50]
VOPD	1.53×1.18	[51]
DVOPD	2×2.23	[51]

Figures 4.16, 4.17, 4.18 and 4.19 illustrate the block diagram and mapping topologies of four NoC applications: MPEG4 decoder (MPEG4), a Multi-Window Displayer (MWD), a Video Object Plane Decoder (VOPD), and an enhanced VOPD application, called DVOPD. We have explained how to make the core txt file in section 4.1.1 as well as the 2D and irregular mapping in section 4.7. We assumed a 90nm technology and a clock frequency of 1GHz. The other specifications of NoC are as follows:

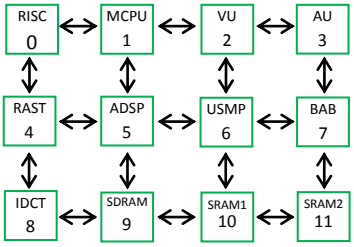
- The size of each flit =31 bits.
- The number of virtual channel =1.
- The depth of FIFO buffer = 2.
- The size of each packet = 5 flits.
- The traffic pattern = application oriented.
- The routing strategy = XY mechanism.



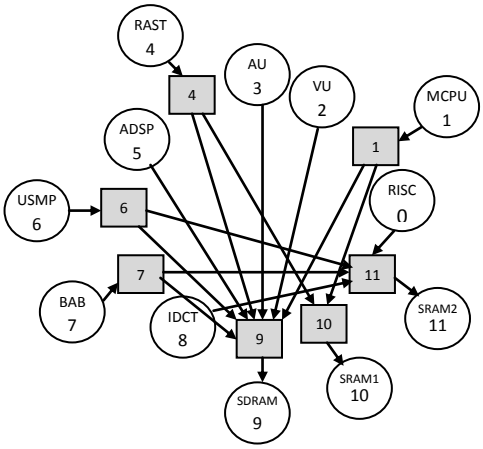
(a) MPEG4 core graph

```
S0D11B500
S1D10B40
S1D9B60
S2D9B190
S3D9B1
S4D10B40
S4D9B600
S5D9B1
S6D9B910
S6D11B670
S7D11B173
S7D9B32
S8D11B250
```

(b) MPEG4 txt core table

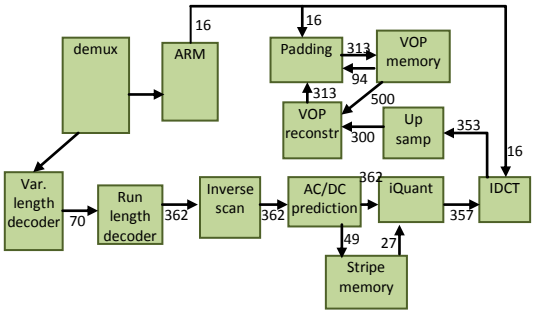


(c) Mesh mapping

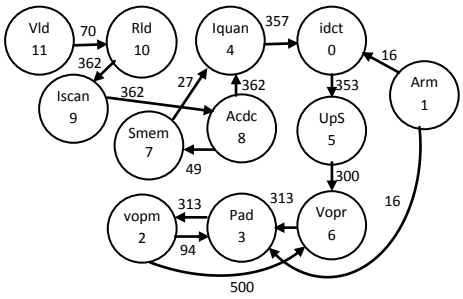


(d) Irregular mapping

Figure 4.16: MPEG4 core graph with communication (in MB/s), txt core table and MPEG4 mapping onto Mesh and application specific topology.



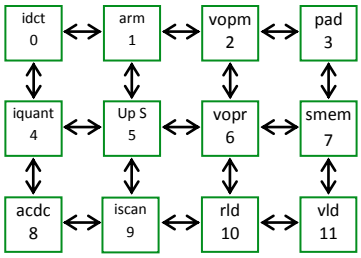
(a) VOPD block diagram



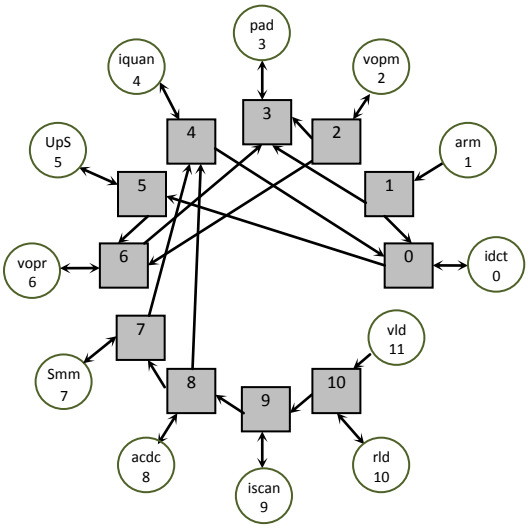
(b) VOPD core graph

S0D5B353
S1D0B16
S1D3B16
S2D3B94
S2D6B500
S3D2B313
S4D0B357
S5D6B300
S6D3B313
S7D4B27
S8D4B362
S8D7B49
S9D8B362
S10D9B362
S11D10B70

(c) Txt core table

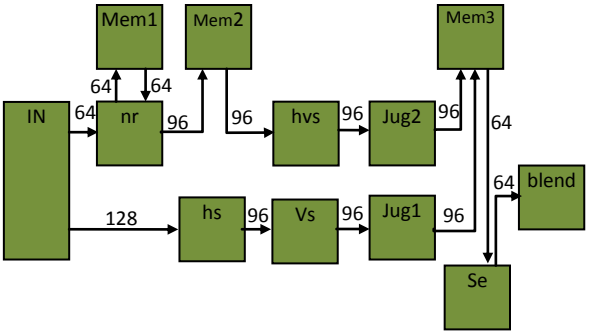


(d) Mesh mapping

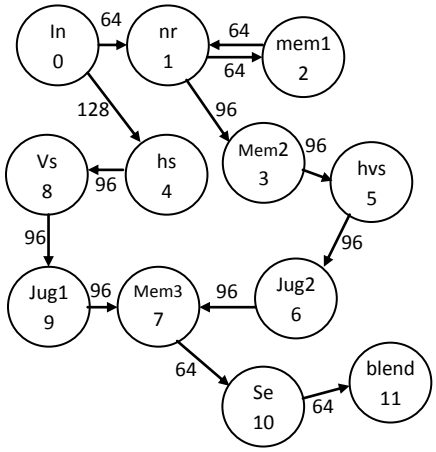


(c) Irregular mapping

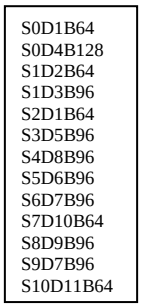
Figure 4.17: VOPD block diagram, core graph with communication (in MB/s), txt core table, and VOPD application mapping onto Mesh and application specific topologies.



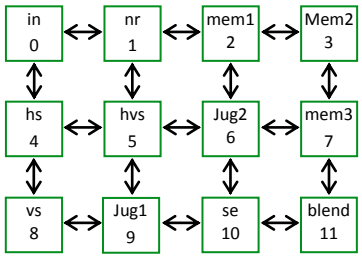
(a) MWD block diagram



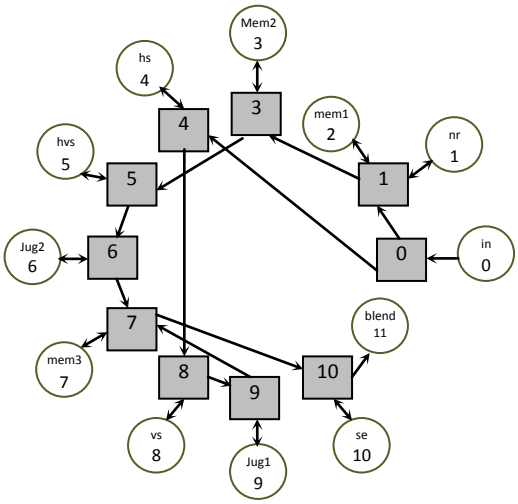
(b) MWD core graph



(c) MWD Txt core table

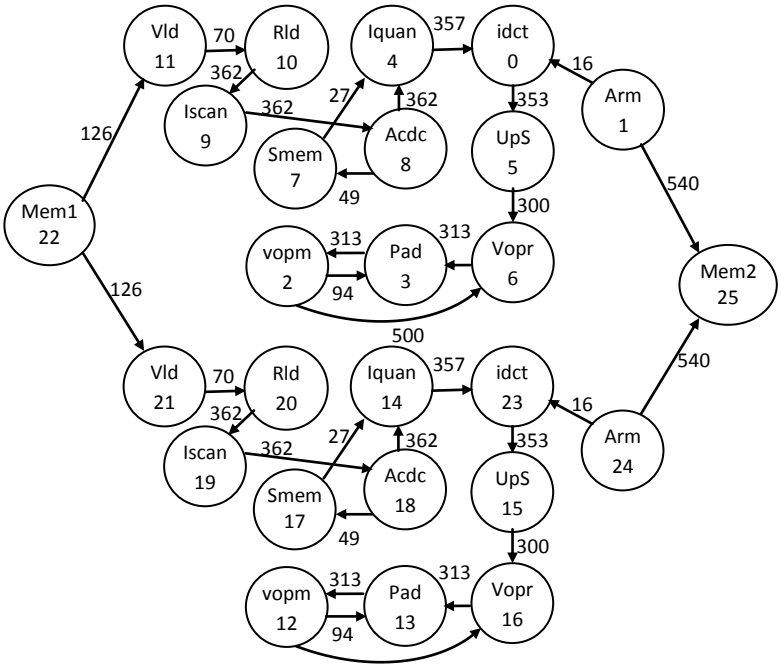


(d) Mesh mapping



(e) Irregular mapping

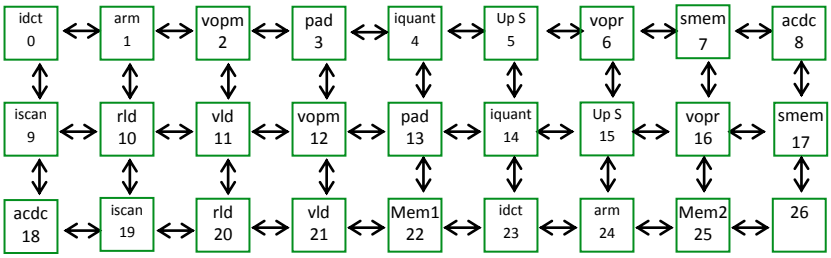
Figure 4.18: MWD signal flow graph, core graph with communication (in MB/s), MWD txt core table and MWD application mapping onto Mesh and application specific topology.



(a) DVOPD block diagram

```
S0D5B353
S1D0B16
S1D25B540
S2D3B94
S2D6B500
S3D2B313
S4D0B357
S5D6B300
S6D3B313
S7D4B27
S8D4B362
S8D7B49
S9D8B362
S10D9B362
S11D10B70
S23D15B353
S24D23B16
S24D25B540
S12D13B94
S12D16B500
S13D12B313
S14D23B357
S15D16B300
S16D13B313
S17D14B27
S18D14B362
S18D17B49
S19D18B362
S20D19B362
S21D20B70
S22D11B126
S22D21B126
```

(b) DVOPD txt core table



(c) Mesh mapping

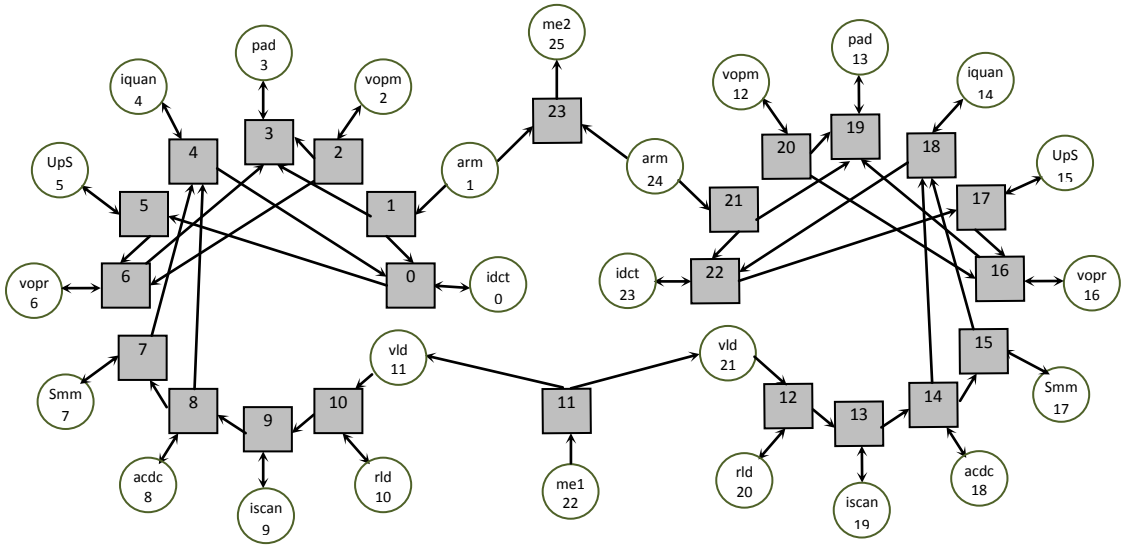


Figure 4.19: DVOPD core graph with communication BW annotated (in MB/s), core txt table and DVOPD mapping onto mesh and application specific topologies

The results are reported in Fig. 4.20 where each histogram is divided into four zones: MPEG4, MWD, VOPD and DVOPD. Each zone contains 3 bars that each bar related to one of mesh, torus and application specific mapping topologies. Each zone contains 3 bars that each bar related to one of mesh, torus and application specific mapping topologies.

Figure 4.20a demonstrates the throughput results of the above mentioned NoC applications in Mesh, Torus and application specific topologies. It is obvious that we cannot predict a specified behaviour for the throughput because the topologies are not the same in each application. In the case of MPEG4 decoder, torus topology is faster and for MWD and VOPD application benchmarks, all three topologies have the same throughput and in the case of VOPD, the application specific topology is faster.

The NoC area results are depicted in Fig. 4.20b. The router areas related to the mesh and torus bars are a function of NoC size. For example, for the MPEG4, MWD and VOPD mapping on torus and mesh, the size of NoC is 3×4. Therefore, the router areas are the same for these applications. However, the router areas of application specific bars are different and depending on the irregular mapping of the application. The area of link is a function of SoC area that consists of the area occupied by the cores and the routers. For example, in torus bars related to MPEG4 and MWD, the link area of MWD bar is bigger than MPEG4 bar because the cores area of MWD application (12) is bigger than MPEG4 application (7.05). Another important point in Fig. 4.20.b is the small amount of application specific bar (i.e. less than 0.2 *mm*) in compared to mesh and torus bars. By considering of the SoC areas of Table 4.5, the NoC areas of torus mapping for VOPD and DVOPD applications are bigger than their application area. However, the application specific chip-area is a good candidate for each area constraints of the design.

The architectural power consumption results of the NoCs are depicted in Fig. 4.20c. These results are an increasing function of changing topologies from application specific to Mesh and then to Torus. This confirms the advantage of application specific over Mesh and Torus topology in terms of minimum resources [4]. The reason for this increase is obvious. The architectural power is a function of structure of NoC, and Torus topology uses more resources than other two topologies. The router architectural power related to the mesh and torus bars are a function of NoC size. For example, for the MPEG4, MWD and VOPD mapping on torus and mesh, the size of NoC is 3×4 therefore the router architectural power are the same for these applications. However, the router architectural powers of application specific bars are different and depending on the irregular mapping of the application. The architectural power of link is a function of the total length of link that estimated by the SoC or NoC system area. The SoC area consists of the area occupied by the cores and the routers. For example, in torus bars related to MPEG4 and MWD, the link architectural power of MWD bar is bigger than the MPEG4 bar because the cores area of MWD application is bigger than the MPEG4 decoder application.

The histogram, which is very important in terms of power for the NoC designer is transactional power consumption. The transactional power consumption is a function of structural and transactional behaviour of NoC. The throughput is a performance metric, which can explain the variation of transaction in NoC, but not in a network with adaptive routing strategy. In other words, when we have deterministic routing strategy, every flit should pass through a specific route. Moreover, in the case of contention the flit has to wait, in spite of adaptive routing where the flit could move to some other routes. Therefore, in the case of mesh, torus and application specific topologies when the routing strategy is deterministic, the higher throughput represents the additional packets (message) passing or more transactions that leads to

additional transactional power consumption. However, the throughput results in Fig. 4.20a are not helpful to explain the transactional power behaviour in Fig. 4.20d as the structural behaviour of each application in Fig. 4.20a is different.

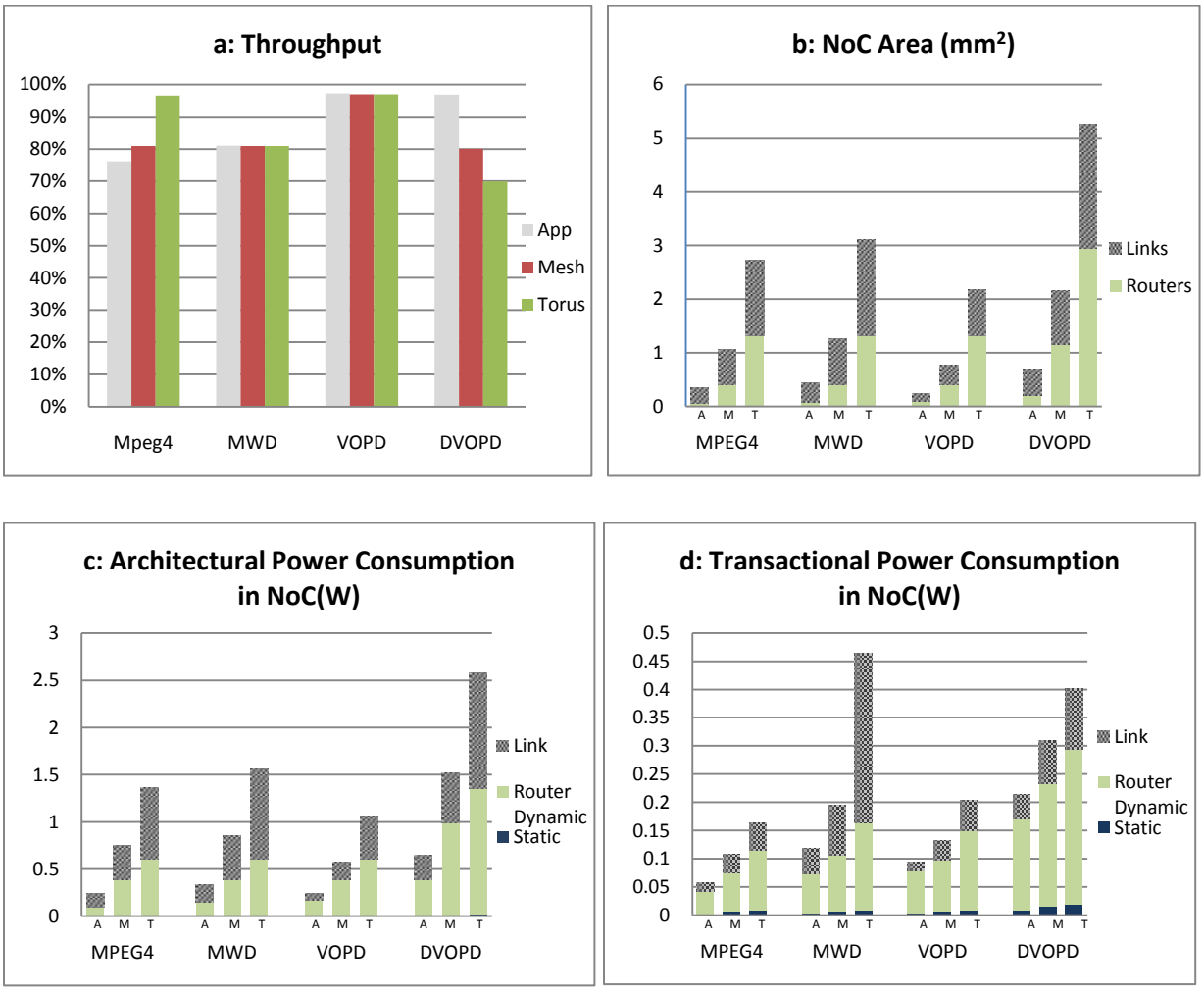


Figure 4.20: NoC synthesized results for the MPEG4, MWD, VOPD and DVOPD applications. Power is expressed in Watts, area in mm^2 and throughput in percent. We used the following notation: M for Mesh, T for Torus, and A for application specific topology.

4.9 Chapter Summary

NoC simulation experiments have been conducted for some well-known routing mechanisms as well as a new LP-routing mechanism. We investigated the locality traffic performance improvement for Network-on-Chip. Moreover, the effect of deterministic and adaptive routing mechanisms in mesh and torus topology has been investigated. It is concluded that the LP-routing mechanism has contention free ability in almost one third of the cases for locality traffic. We also did investigate for application specific topologies such as MPEG-4 Decoder and AV Benchmark applications. It is observed that LP routing is more effective when the network system has extra sharing resources such as links, channels and techniques and when there is no PTMP contention. For the second set of experiments, we selected a number of SoC applications and traded off their results by considering further on power and area metrics. We explained the relation of throughput with transactional power consumption and observed that the throughput can explain the behaviour of transactional power for deterministic routing strategy.

Chapter 5

Conclusions

In this work, we have presented the development of a SystemC base NoC simulator, FAANOS, and conducted experiments on various metrics of NoC design such as performance, power and chip-area. We described the concept, architecture and configuration of NoC as well as introduced the detailed structure and parameters of our simulator. Different kinds of traffic patterns have been employed in the simulation of NoC. We broke down the concept of performance to four characteristics such as throughput, latency, packet drop and link utilization that play an important role to choose an effective design. We did a brief survey on some previous works and found that they only output architectural power estimation of NoCs. Instead, our work produces both transactional power estimation and architectural power estimation, where the first one is very important parameter as compared to the second one in designing an NoC system. We have presented a comparison analysis between the capability of our simulator and two contemporary NoC simulators: NIRGAM and NOSTRUM. We highlighted the capability of our simulator on different aspects and demonstrated its superior features. The structure of FAANOS was analyzed, and the cycle-accurate models of the hardware part of simulator were described.

We demonstrated the development of our NoC simulator from a simple on-chip network to a more complex and automatic NoC simulator. Then we proposed an analytical model for the power and area of different modules of an NoC such as FIFO buffer, arbiter, crossbar, and link. We have presented different architectural models of each module. Then we have decomposed them into gate level models and presented the technological and analytical parameters accompanied by the related formula. A test methodology for early design of NoC was presented, and based on it two sets of experiments has been conducted for some characteristics of the NoC. In the first set of experiment, it is argued that locality traffic has great performance improvement in network on chip. Moreover, the effect of deterministic and adaptive routing mechanism in mesh and torus topology has been investigated. We concluded that the LP routing mechanism has contention free ability in about one third of the cases of locality traffic when there is communication between one node to another node. We also did perform simulation experiments for application specific topologies for MPEG-4 Decoder and AV Benchmark application. We concluded that the LP routing is more effective when the network has more extra resources such as links, channels and techniques (being regular than being irregular), and when there is no PTMP contention in NoC (the instant rate of generation of flits is less than or equal to the instant capacity of incoming flits).

In the second set of experiments, we conducted experiments on some NoCs by applying two kinds of traffic patterns: dummy and application oriented pattern. The relationship between the throughput and transactional power consumption is explored and it is concluded that the throughput can explain the behaviour of transactional power when the routing strategy is deterministic. The SoC technology is expanding very fast with new applications in consumer electronics. So far the number of IP used in SoC is in the range of tens, and without a doubt it is

going to increase and sometime in the future reach millions IP. This vast development in SoC needs more research especially in the NoC area. One layer of NoC that has not been researched fully in the past is the topology generation. Topology generation is mapping an SoC application on the different topologies including regular or irregular to reach an optimized system. It is obvious that our work in this thesis can be a small part of topology generation. Therefore, the future work is going to research this part of NoC technology.

References

[1] Gul N. Khan, V. Dumitriu,“ A modeling tool for simulating and design of on-chip network systems,” Original Research Article *Microprocessors and Microsystems*, vol *34*, No. 2-4, pp. *84-95*, March-June 2010.

[2] Tobias Bjerregaard and Shankar Mahadevan*, ”*A Survey of Research and Practices of Network-on-Chip*,”* ACM Computing Surveys, Technical University of Denmark, vol. 38, No. 1, June 2006.

[3] L. Benini and G. De Micheli, “Networks on chips: A new SoC paradigm,” IEEE Computer, vol. 35, No. 1, pp. 70–78, Jan. 2002.

[4] Masoud O. Gharan and Gul N. Khan*,”*Flexible simulation and modeling for 2D topology NoC system design” IEEE CCECE 2011: Symposium on Computers, Software and Applications, Niagara Falls, Canada, May, 2011.

[5] http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/PowerCompiler.aspx , accessed 1 September 2011.

[6] H. Wang, X. Zhu, L.-S. Peh and S. Malik, ”Orion: A Power-Performance Simulator for Interconnection Networks,” Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture, Istanbul, Turkey, pp. 294-395, 2002.

[7] D. Brooks, V. Tiwari and M. Martonosi, ”Wattch: A Framework forArchitectural-Level Power Analysis and Optimizations,” Proceedings of the 27th International Symposium on Computer Architecture, Vancouver, Canada, pp. 83-94, 2000.

[8] K. Goossens, M. Bennebroek, J.Y. Hur, M.A. Wahlah, “Hardwired networks on chip in FPGAs to unify functional and configuration interconnects,” IEEE International Symposium on Networks-on-Chip, *Newcastle, United Kingdom,* pp. 45–54, 2008.

[9] M.B. Stensgaard, J. Sparsø, “ReNoC: A network-on-chip architecture with reconfigurable topology,” IEEE International Symposium on Networks-on-Chip, Newcastle, United Kingdom, pp. 55–64, 2008.

[10] S. Kumar, A. Jantsch, J.P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, “A network on chip architecture and design methodology,” Proceedings IEEE Computer Society Annual Symposium on VLSI, Pittsburgh, Pennsylvania, pp. 105 – 112, 2002.

[11] ATIS committee PRQC. "network topology". *ATIS Telecom Glossary 20011*, accessed 1 September 2011.

[12] G.N. Khan and V. Dumitriu, “Simulation environment for design and verification of Network-on-Chip and multi-core systems,” IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems, London, England, pp. 1 – 9, 2009.

[13] Jantsch Axel and Tenhunen Hannu, “ Network on Chip,” Royal Institute of Technology, Stockholm Springer, book, VIII, 303 p, 2003.

[14] V. Dumitriu and G.N. Khan, “Throughput-Oriented NoC Topology Generation and Analysis for Performance SoCs,” IEEE Transactions on VLSI Systems, vol. 17 , No. 10, pp. 1433 - 1446, 2009.

[15] L.G. Roberts, “The evolution of packet switching,” Proceedings of the IEEE, vol. 66 , No. 11, pp. 1307 - 1313, 1978.

[16] F. Verbeek, J. Schmaltz, “A fast and verified algorithm for proving Store-and-Forward networks Deadlock-Free,” 19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Ayia Napa, Cyprus, pp. 3 - 10, 2011.

[17] A. Roca, J. Flieh, F. Silla, J. Duato,” VCTlite: Towards an efficient implementation of virtual cut-through switching in on-chip networks,” International Conference on High Performance Computing (HiPC), Goa, India, pp. 1 – 12, 2010.

[18] Ankur Agarwal, Cyril Iskander, and Ravi Shankar, “Survey of Network on Chip (NoC) Architectures & Contributions,” Journal of Engineering, Computing, and Architecture, vol. 3, No. 1, 2009.

[19] Ge-Ming Chiu, “The odd-even turn model for adaptive routing,” IEEE Transactions on Parallel and Distributed Systems, vol. 11, No. 7, pp. 729 – 738, 2000.

[20] Wang Zhang, Ligang Hou, Jinhui Wang, Shuqin Geng, and Wuchen Wu, “Comparison research between XY and Odd-Even routing algorithm of a 2D 3×3 mesh topology Network-on-Chip,” WRI Global Congress on Intelligent Systems, Xiamen, China, vol. 3, pp. 329 – 333, 2009.

[21] Yoshio Turner and Yuval Tamir, “Deadlock-free connection-based adaptive routing with dynamic virtual circuits,” Journal of Parallel and Distributed Computing, vol. 67, No. 1, pp. 13-32, January 2007.

[22] J. Duato, O. Lysne, R. Pang, T.M. Pinkston, "Part I: A theory for deadlock-free dynamic network reconfiguration," IEEE Transactions on Parallel and Distributed Systems, vol. 16, No. 5, pp. 412– 427,2005.

[23] G. Michelogiannakis, D. Sanchez, W.J. Dally, C. Kozyrakis," Evaluating bufferless flow control for On-chip networks," Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS), Grenoble, France, pp. 9 – 16, 2010.

[24] P.N. Parakh, R.B. Brown, K.A. Sakallah," Congestion driven quadratic placement," Proceedings of Design Automation Conference, San Francisco, CA, pp. 275 – 278, 1998.

[25] A.B. Kahng, Bin Li, Li-Shiuan Peh, and K. Samadi, "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Nice, France, pp. 423 – 428, 2009.

[26] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The Alpha 21364 network architecture," IEEE Micro, vol. 22, No. 1, pp. 26 – 35, 2002.

[27] IBM. IBM InfiniBand 8-port 12x switch. http://www3.ibm.com/chips/products/infiniband/.

[28] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," IEEE Micro, vol. 27, No. 5, pp. 51-61, 2007.

[29] D. A. IIitzky, J. D. Hoffman, A. Chun, and B. P. Esparza, "Architecture of the Scalable Communications Core's Network on Chip,"IEEE Micro, vol. 27, No. 5, pp. 62-74, 2007.

[30] Ivan E. Sutherland and Jo Ebergen, "Computers without Clocks," Scientific American Magazine, pp. 62-69, August 2002.

[31] R. Ho, K. Mai, and M. Horowitz, "The Future of Wires," Proceedings IEEE, vol. 89, No. 4, pp. 490-504, Apr. 2001.

[32] A. Banerjea and S. Keshav, "Queueing delays in rate controlled ATM Networks," IEEE INFOCOM, San Francisco, CA , pp. 547 - 556, 1993.

[33] Lu Zhonghai and A. Jantsch, "Traffic configuration for evaluating networks on chips," Proceedings of the Fifth International Workshop on System-on-Chip for Real-Time Applications, Banff, Alberta, Canada, pp. 535 – 540, 2005.

[34] X. Chen and L.-S. Peh, "Leakage power modeling and optimization in interconnection networks ," Proceedings of the International Symposium on Low Power Electronics and Design, Seoul, Korea, pp. 90-95, 2003.

[35] S. Thoziyoor, N. Muralimanohar, J. H. Ahn and N. P. Jouppi, "CACTI 5.1," *Technical Report HPL-2008-20*, HP Laboratories, 2008.

[36] H. Yoshida, D. Kaushik and V. Boppana, "Accurate Pre-Layout Estimation of Standard Cell Characteristics," Proceedings of the 41st Design Automation Conference, San Diego, CA, pp. 208-211, 2004.

[37] Bo Zhang, T.S.E. Ng, A. Nandi, R.H. Riedi, P. Druschel, and Guohui Wang," Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space," IEEE/ACM Transactions on Networking, vol. 18 , No. 1, pp. 229 – 242, 2010.

[38] Jie Wu, "A deterministic fault-tolerant and deadlock-free routing protocol in 2D meshes based on odd-even turn model," ACM Proceedings of the 16th international conference on Supercomputing, New York, NY, USA, June 2002.

[39] A. Banerjee, R. Mullins and S. Moore, "A Power and Energy Exploration of Network-on-Chip Architectures," Proceedings of the First International Symposium on Networks-on-Chip, Princeton, New Jersey, pp. 163-172, 2007.

[40] A. Bona, V. Zaccaria, and R. Zafalon, "System Level Power Modeling and Simulation of High-End Industrial Network-on-Chip," Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Paris, France, pp. 318-323, 2004.

[41] S. Bhat, "Energy Models for Network-on-Chip Components," M.S. Thesis, Dept. of Mathematics and Computer Science, Royal Institute of Technology, Eindhoven, 2005.

[42] Wang Zhang, Wuchen Wu, Lei Zuo, and Xiaohong Peng, "The buffer depth analysis of 2-Dimension mesh topology Network-on-Chip with Odd-Even routing algorithm," International Conference on Digital Object Identifier, Wuhan, China, pp. 1 - 4, 2009

[43] http://www.systemc.org , accessed 1 September 2011.

[44] H. Wang, L. Peh, and S. Malik, " A Power Model for Routers: Modeling Alpha 21364 and Infiniband Routers," IEEE Micro, vol. 23, No. 1, pp. 27-35, 2003.

[45] Y. Hoskote, S. Vangal, A. Singh, N. Borkar and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," IEEE Micro, vol. 27, No. 5, pp. 51-61, 2007.

[46] M. B. Taylor *et al.*, "Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams," Proceedings of the 31st International Symposium on Computer Architecture, München, Germany, pp. 2-13, 2004.

[47] D. E. Duarte, N. Vijaykrishnan and M. J. Irwin, "A Clock Power Model to Evaluate Impact of Architectural and Technology Optimization," IEEE Transactions on VLSI Systems, vol. 10, No. 6, pp. 844-855, 2002.

[48] S.Heo and K. Asanovic, "Replacing Global Wires with an On-Chip Network: A Power Analysis," Proceedings of the International Symposium on Low Power Electronics and Design, San Diego, California, pp. 369-374, 2005.

[49] S. Murali and G. De Micheli, "Sunmap: A tool for automatic topology selection and generation for NoCs," Proceeding Design Automation Conf, San Diego, Ca, pp. 914-919, 2004.

[50] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. D. Micheli, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," IEEE Trans. on Parallel and Distributed Systems, vol. 16, No. 2, pp. 113–129, Feb. 2005.

[51] A. Pullini, F. Angiolini, P. Meloni, D. Atienza, S. Murali, L. Raffo, G. D. Micheli, and L. Benini, "NoC Design and Implementation in 65nm Technology," Proceedings of the First International Symposium on Networks-on-Chip, Princeton, New Jersey, pp. 273 – 282, 2007.