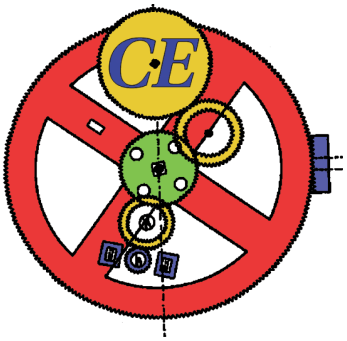


**Computer Engineering**
Mekelweg 4,
2628 CD Delft
The Netherlands
**http://ce.et.tudelft.nl/**

2014

# MSc THESIS

# NoC characterization framework for design space exploration

**Sriram Prakash Adiga**

**CE-MS-2014-07**

## Abstract

A Network on Chip (NoC) is considered as the interconnect architecture for the future Multi Processor System on Chip (MPSoC). It should provide the required quality of service in terms of predictable guaranteed as well as best effort services to meet the application constraints. The application traffic patterns applied on a NoC is hard to predict due to dynamics of the application. A characterization framework is required for design space exploration of NoC architectures. This framework should be able to evaluate NoC performance for a set of applications.

In this thesis we propose a NoC characterization framework for design space exploration called NoCExplorer. NoCExplorer is a cycle accurate simulator, developed using SystemC and VHDL. It supports individual blocks in the NoC to be synthesized or replaced with register transfer level VHDL code for mixed language simulations. NoCExplorer provides a rich set of performance parameters to analyze a NoC and can generate heat maps for visual representation of link utilization and router congestion in the NoC. It was correlated with many journals and publications to prove the correctness of the results. Results from the experiments in this thesis show that the NoC performance is impacted by cycle interval between flit injection into the network. An increase in the interval between the flit injection within the packet significantly increases the average latency of the NoC. The ratio of increase in latency varies for different NoCs under a set of synthetic traffic patterns. NoCExplorer provides capability to map custom application over the NoC to evaluate its performance. A Digital Audio Broadcasting (DAB) application transaction model was mapped on many NoCs. We are able to assess the performance of many NoCs and visualize the congestion and link utilization. Stressing the NoC by reducing its clock frequency by a factor of 100x of core operating frequency, one of the spidergon based NoC with *across first* routing algorithm showed best results for the given application constraints.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

# NoC characterization framework for design space exploration

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Sriram Prakash Adiga
born in Manipal, India

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

# NoC characterization framework for design space exploration

by Sriram Prakash Adiga

**Abstract**

A NoC is considered as the interconnect architecture for the future MPSoC. It should provide the required quality of service in terms of predictable guaranteed as well as best effort services to meet the application constraints. The application traffic patterns applied on a NoC is hard to predict due to dynamics of the application. A characterization framework is required for design space exploration of NoC architectures. This framework should be able to evaluate NoC performance for a set of applications.

In this thesis we propose a NoC characterization framework for design space exploration called NoCExplorer. NoCExplorer is a cycle accurate simulator, developed using SystemC and VHDL. It supports individual blocks in the NoC to be synthesized or replaced with register transfer level VHDL code for mixed language simulations. NoCExplorer provides a rich set of performance parameters to analyze a NoC and can generate heat maps for visual representation of link utilization and router congestion in the NoC. It was correlated with many journals and publications to prove the correctness of the results. Results from the experiments in this thesis show that the NoC performance is impacted by cycle interval between flit injection into the network. An increase in the interval between the flit injection within the packet significantly increases the average latency of the NoC. The ratio of increase in latency varies for different NoCs under a set of synthetic traffic patterns. NoCExplorer provides capability to map custom application over the NoC to evaluate its performance. A DAB application transaction model was mapped on many NoCs. We are able to assess the performance of many NoCs and visualize the congestion and link utilization. Stressing the NoC by reducing its clock frequency by a factor of 100x of core operating frequency, one of the spidergon based NoC with *across first* routing algorithm showed best results for the given application constraints.

**Laboratory** : Computer Engineering
**Codenumber** : CE-MS-2014-07

**Committee Members** :

| | |
|---|---|
| **Advisor:** | Zaid Al-Ars, CE, TU Delft |
| **Chairperson:** | Koen Bertels, CE, TU Delft |
| **Member:** | Marco Zuniga, CS, TU Delft |
| **Member:** | Eelke Strooisma, Recore Systems |

*Dedicated to my grandmother*

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**NoC** Network on Chip

**MPSoC** Multi Processor System on Chip

**SoC** System on Chip

**DAB** Digital Audio Broadcasting

**SDF** Synchronous Data Flow

**ASIP** Application Specific IP

**QoS** Quality of Service

# Acknowledgements

When I decided to do masters after a long stint in the industry, I had doubts if I can be a student once again. Looking back at eventful 2 years I spent in Netherlands, I feel it was one of the right decisions in my life. This could not have been done without the support of my friends and family.

The work presented in thesis was outcome of research performed at Recore Systems, Netherlands. I would like to thank Gerard and Gilles for providing me an opportunity and treating me on par as an employee at the company. It was an enriching experience to operate in a small company with lot of flexibility and open to new ideas. This was one of the key factors in shaping up of my thesis work as part of my MSc program in Embedded Systems.

I would like to thank Eelke, Yifan and Sebastien at Recore Systems for spending time every week to guide and help me with my thesis work. I learned a lot on approach to create a system, critical thinking and documentation.

I would also like to thank Dr Zaid-Al-Ars for guiding me with the thesis and providing me valuable inputs at various stages of my work. This document which I am proud of would not have been possible without the critical inputs and suggestion from Eelke and Zaid.

Finally I would like to thank all my friends at *Timepass Inc* in Delft and Anantha for wonderful time we spent in 2 years. I would like to thank my parents for having faith in me to purse studies away from home.

Sriram Prakash Adiga
Delft, The Netherlands
June 26, 2014

# 1 Introduction

In recent years, emerging heterogeneous Multi Processor System on Chip (MPSoC) are providing potential to execute single or several complex applications concurrently. A MPSoC design flow as explained in [1] starts from customer requirements or marketing requirements which would specify and describe the overall application at abstract level. The requirements are converted into functional models in a high level language such as C/C++. Based on the high level functional models, designers perform hardware and software partitioning. Architecture explorations are done to select best possible hardware and software components for the functional models. The architectural model defines computational blocks which have to communicate with each other. The communication model defines the interconnect architecture of a System on Chip (SoC). The model responsibility is to ensure single or multiple co-existing data streams are correctly routed from source components to destination components. The model should also provide latency and bandwidth guarantees to meet the application constraints. To meet the application constraints, various interconnect architectures for a SoC have been developed like simple bus, segmented bus, pipelined bus and others. Due to various constraints in the mentioned interconnect architectures, Network on Chip (NoC) is considered to be interconnect architecture for the future MPSoC.

## 1.1 Thesis motivation

A NoC provides the backbone communication link between components in a MPSoC. Hence. it should provide the required quality of service in terms of predictable guaranteed service and best effort services to meet the application constraints. The application traffic patterns applied on a NoC is hard to predict due to the dynamics of the application. A characterization framework is required for design space exploration to consider various types of NoC architectures to evaluate their performance for a set of applications. It should be able to consider combination of various techniques and architectures like routers, routing algorithms, topologies, network interfaces and others to generate a NoC. The framework should be capable of generating traffic patterns for network simulation. The most realistic traffic patterns are application driven, which is mapped on the NoC. It is hard to predict the application to be mapped on the NoC, hence synthetic traffic pattern can simulate the demands expected by the NoC. The framework should be capable of providing performance of characteristics of various NoC architectures for applied traffic patterns. This will help designers to choose appropriate NoC which meets the constraints as interconnect architecture for there MPSoC.

## 1.2 Thesis contributions

In the thesis we propose a NoC characterization framework for design space exploration called NoCExplorer. The NoCExplorer is a cycle accurate simulator, developed using SystemC and VHDL. It supports various techniques and architecture block such as routing algorithm, virtual channels, topologies and others to generate a NoC. Python based scripting is used to retrieve, interpret and present the performance of the NoC in a flexible, user customizable way for the designers.

A comparison with available open source simulators has been done to show the added value of NoCExplorer. The NoCExplorer was correlated with multiple journals and publications to prove the correctness of the results. A Digital Audio Broadcasting (DAB) application transaction model was used to do design space exploration of NoC using NoCExplorer. The DAB application was mapped, simulated and results were analyzed on selected set of NoC architectures using NoCExplorer.

## 1.3 Thesis outline

The thesis has been organized into multiple chapters to explain the motivation, design and results of NoCExplorer.

In Chapter 2, we explain the reason to use a NoC over bus based interconnect architecture for a MPSoC. After that we explain the techniques and architecture used inside the NoC which are router architecture, network interface, topology, routing algorithms, flow control and others. We also provide methods and parameters to assess the performance of a NoC. Then we present related work on NoC simulators. We explain capabilities and limitations of each NoC simulator.

NoCExplorer design is explained in Chapter 3. We start with overall architecture of the design and explain each block in detail.

Capabilities and evaluation of NoCExplorer is shown in Chapter 4. The results of NoCExplorer are compared with few journals and publications to prove the correctness of the results.

In Chapter 5, we explain the DAB application transaction model. After that we show how application transaction model can be applied on NoCExplorer. Using NoCExplorer we show different design space exploration done for various NoC architectures and provide optimal NoC to use for given constraints.

Conclusion and recommendations for future work of NoCExplorer are provided in Chapter 6.

# Network on chip and related work

# 2

In this chapter we explain the need for Network on Chip (NoC) by comparing NoC and bus based interconnect architectures for a System on Chip (SoC). After that we explain the techniques and architecture of a NoC in detail like topology, router architecture, network interface, routing algorithm and others. We also explain performance evaluation of NoC, usage of simulators and types of simulators available.

## 2.1 Bus based interconnect architecture

A bus based communication architecture as shown in Figure 2.1 is one of the communication models used in current Multi Processor System on Chip (MPSoC)s. Components are interconnected by a bus. Major characteristics of the bus architecture are *a*) one transaction at a time *b*) central arbiter *c*) limited bandwidth *d*) synchronous *e*) low cost [2].

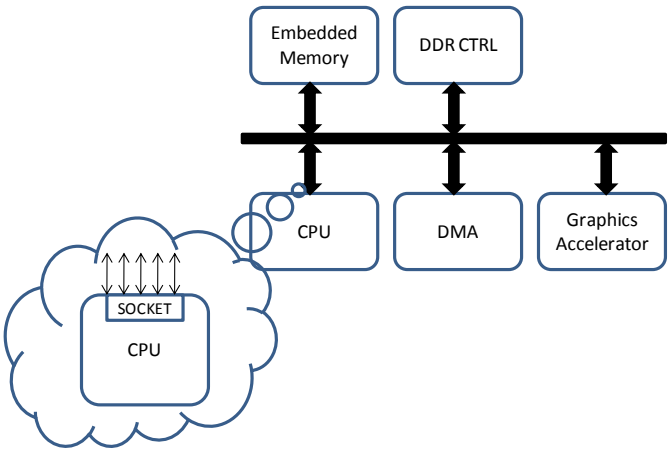
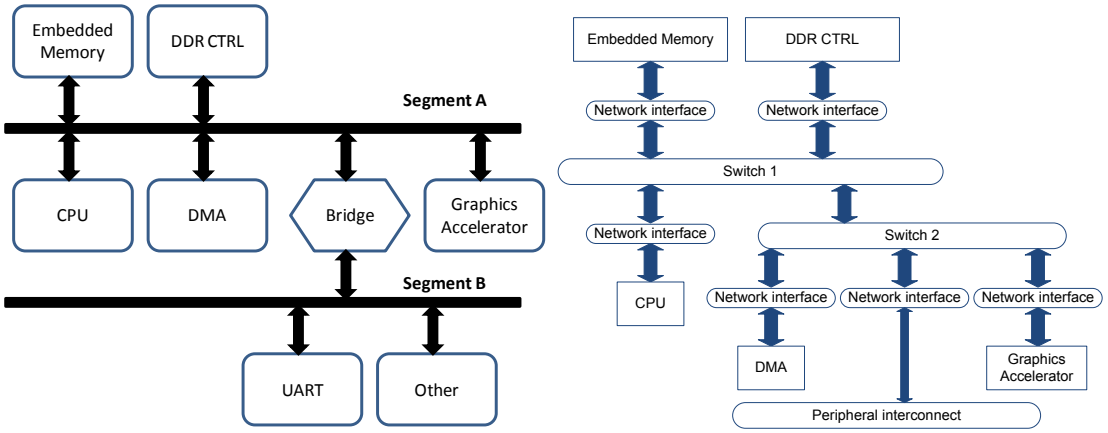



Figure 2.1: Example of an on chip bus

As the number of cores and IP's integrating in the MPSoC increase, a simple bus architecture became a bottleneck for communication. It is due to sharing of aggregate bandwidth for all components and increases arbitration delay. Hence, a few advanced bus architecture such as segmented bus, pipeline packetized multistage crossbar has been mentioned in [2] and as shown in Figure 2.2. Some of the key features of the advanced bus architecture are *a*) versatile compared to simple bus architecture *b*) pipeline capability *c*) burst transfer *d*) split transactions *e*) overlapped arbitration *f*) transaction preemption and resumption *g*) transaction reordering.

The next generation MPSoCs keep pushing for higher operating frequency, increase in




(a) Segmented bus (b) Pipeline packetizied multistage crossbar

Figure 2.2: Examples of advanced bus architectures

integration of memory, cores and IP's which we will call as nodes. Hence it would increase bandwidth required for communication between the nodes. The mentioned bus architecture is not scalable since bandwidth is fixed at design time and shared which means throughput decreases with increase in number of nodes. Implementation of pipelining is complex and central arbitration per layer or bus is required. So bus architecture will become communication bottleneck in the MPSoCs. Hence in the next section we will explain a new communication paradigm called NoC.

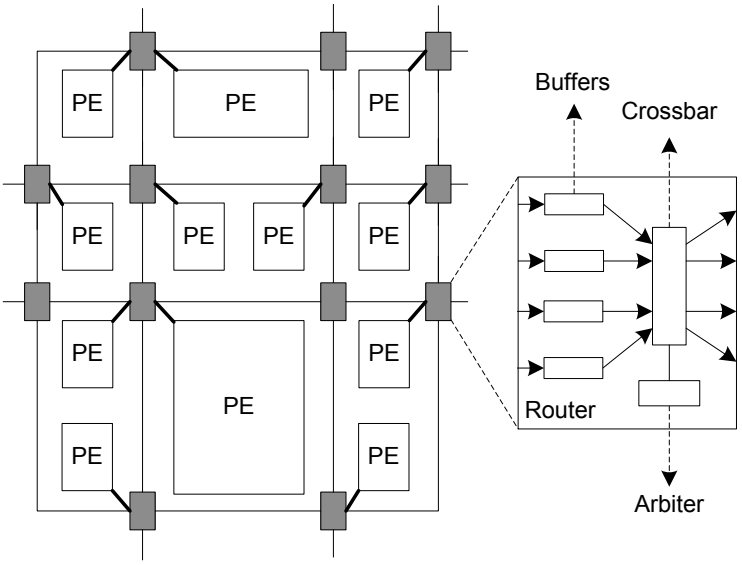



Figure 2.3: An example NoC interconnect

A NoC is considered to be viable interconnect architecture for future MPSoC design [1]. It uses packets to route data from source to destination via a network fabric as shown in Figure 2.4. The data is converted into packets and packets are divided into

flits such as head flit, body flit and tail flit. Head and body flit contains information required for the packet to traverse through the network fabric like direction of packet, buffer to used and others. Body flits carries the data. The network fabric consists of

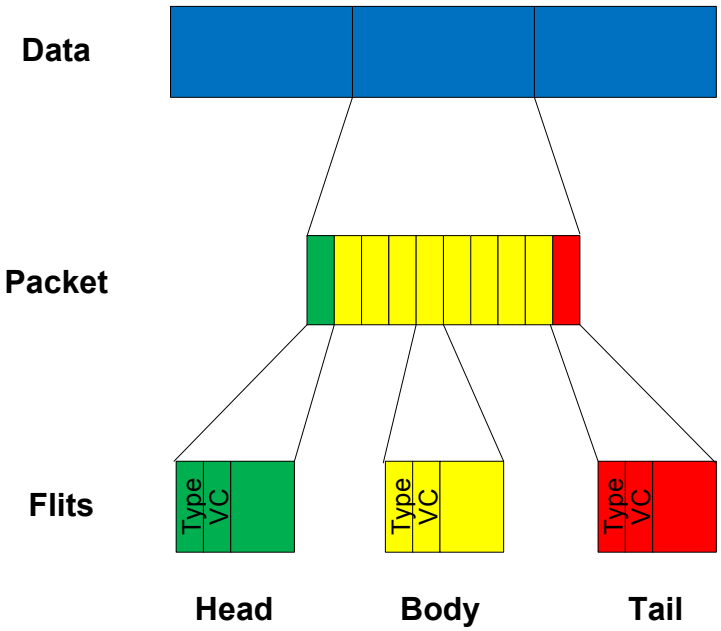



Figure 2.4: Packet and flit format used in NoC

routers and interconnection links (wires) as shown in Figure 2.3. A NoC interconnection architecture consists several nodes connected together by network interface, routers and wires. A network interface is at boundary of node connected to a router. It converts the data generated by the node into flits and packets. A router accepts the packets generated by network interface or other routers connected to it. It has buffers at the input or output based on the router architecture to store the packets received. The packets are transported to destination link via the crossbar switch based on address mentioned in the packets. An arbiter is used to determine priority for a packet to be serviced if multiple packets from different source requires same output link. Thus packets traverse multiple links and hop multiple routers in the NoC from source to destination node. The data is extracted for received packets at destination node by network interface.

With increase in nodes in the network, NoC link speed does not get affected. And there is aggregate growth in the bandwidth due to inherent structure and design of the NoC. It has built-in capability to pipeline transfers of packets in network interface and routers. The arbitration of packets is distributed across network interfaces and routers which can be classified into various levels of abstraction layers. In Table 2.1 we summarize main features of NoC over bus architecture. But disadvantage of NoC over bus architecture is overhead of area, power and delay in routers.

Table 2.1: Comparison of NoC and bus architecture [2]

| **NoC** | **Bus** |
|---|---|
| Aggregate growth in bandwidth | Bandwidth is limited, shared |
| Link speed unaffected by number of nodes | Speed goes down as nodes increases |
| Built in pipeline | Pipelining is tough |
| Distributed arbitration | Central arbitration per layer |
| Separate abstraction layers | No layers of abstraction |
| Performance guarantee is complex to assess | Fairly simple implementation |
| Extra delay in routers | |
| Area and power overhead | |

## 2.2 NoC architecture

A basic NoC architecture consists of various techniques and blocks connected together to form interconnect architecture for a SoC. Main aspects of the NoC architecture are

- routing algorithm
- flow control
- router
- topology
- network interface.

Peh *et al.* [3], Dally *et al.* [4] and others have written in detail about these parts and properties used in the NoC. In next section will briefly discuss these parts and properties which will be relevant to this thesis.

### 2.2.1 Routing algorithm

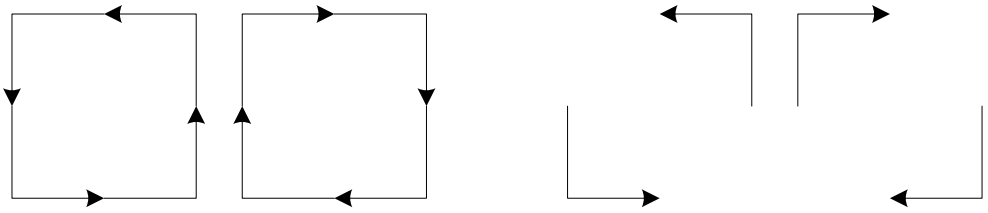
The goal of a routing algorithm is to assure low latency and high throughput in the NoC. It can be achieved by even traffic distribution, avoid hot spots and minimize contention for packets. Major issue in implementation of routing algorithms are deadlock, livelock, starvation and traffic distribution to achieve better Quality of Service (QoS). Deadlock happens when two or more packets in the network are waiting each other to be routed forward. Livelock happens when packet keep spinning around its destination without ever reaching it. Starvation happens when different priorities are used with lower priority packets would never reach the destination because there is always a higher priority packet. The mentioned issues in routing algorithms can be resolved with certain techniques in the algorithm.

Different types of routing algorithm has been highlighted in [3] which can be broadly classified into three categories

- Deterministic

- Oblivious
- Adaptive

Deterministic routing algorithms are based on dimension order routing algorithm. Dimension order routing are routing algorithms which can be described by which turns are permitted. In Figure 2.5a we illustrate all possible turns in a mesh and torus based network. Example dimension order based XY routing algorithm allows limited set of permissible turns in the network as shown in Figure 2.5b. In oblivious routing algo-



(a) All turns (b) X-Y turns

Figure 2.5: Possible routing turns in mesh and torus based NoC

rithm paths are selected without the regard to network congestion. In adaptive routing algorithm paths are selected based on network congestion. In Table 2.2 and Table 2.3 we have described different routing algorithms and its features. Each routing algorithm listed in the table has been explained in detail by Rantala *et al.* [5], Coppola *et al.* [2] and Peh *et al.* [3]. In NoCExplorer we are supporting some of the routing algorithms mentioned in the table, which would be explained in next chapter. For future work, NoCExplorer should support all these routing algorithms with support to recover from deadlock, livelock and starvation.

### 2.2.2 Flow control

Flow control determines the allocation of resources for network buffers and links. It controls assignment of packets to links and buffers and how these resources are shared among many packets using the network. Flow control should aim to decrease the latency of the packet by keeping resource allocation over head as minimum as possible. It should improve network throughput by enabling effective sharing of buffers and links across packets [3]. Agarwal *et al.* [6] have mentioned various types of buffered flow control techniques

- Credit based flow control
- Handshake signals
- ACK or NACK flow control
- T-Error flow control

Table 2.2: Comparison of deterministic and oblivious routing algorithms [5][3][2]

| **Algorithm** | **Outlines** | **Deadlock** | **Livelock** |
|---|---|---|---|
| Dimension order | Routing in one dimension at a time | - | - |
| XY | Routing first in X and then in Y dimension | - | - |
| Pseudo adaptive XY | Partly adaptive XY routing | - | - |
| Across first/last | Route across the link first/last | + | - |
| Surrounding XY | Partly adaptive XY routing | - | - |
| Turn model | Few turns forbidden | + | - |
| Valiant's Random | Partly stochastic | - | - |
| Probabilistic flood | Stochastic | + | + |
| Random walk | Stochastic | + | + |
| Source | Sender determines the route | - | - |
| Directed flood | Stochastic | + | + |
| ALOAS | Application of source routing | - | - |
| Topology adaptive | Reprogrammable routing table | - | - |
| Destination tag | Routers determine the route | - | - |

+ : Possible, - : Not possible

Table 2.3: Comparison of adaptive routing algorithms [5][3]

| **Algorithm** | **Outlines** | **Deadlock** | **Livelock** |
|---|---|---|---|
| Minimal adaptive | Shortest path routing | - | - |
| Fully adaptive | Congestion avoidance | - | - |
| Congestion look ahead | Congestion avoidance | - | - |
| Turnaround or Turnback | Routing in butterfly and tree networks | - | - |
| Turn back when possible | Routing in tree networks | - | - |
| IVAL | Improved turnaround routing | - | - |
| 2TURN | Slightly determined | - | - |
| Q | Statistics based routing | + | + |
| Odd Even | Turn model | - | + |
| Hot potato | Routing without buffer memories | + | + |

+ : Possible, - : Not possible

In credit based flow control an upstream router or node keeps count of data transfers. Available free slots are called credits. When the data is transmitted to next stage or consumed the credit is sent back to the source. In handshake signal based protocol a valid signal is sent when flit is transmitted. The receiver sends a valid signal once data is consumed. ACK or NACK flow control is similar to handshake signal protocol but copy of data flit is kept in buffer till ACK signal is received. If NACK signal is received the data is scheduled for retransmission. This flow control is used in XPIPES implementation

[7]. The T-Error flow control is a complex flow control mechanism compared to other flow control mechanisms. It is targeted for enhancing the performance at the cost of reliability.

### 2.2.3 Router

A router is considered to be the heart of NoC. The design requires to meet latency and throughput requirements with tight area and power constraints. A router determines hop delay and influence network latency. An example credit based virtual channel router architecture is shown in Figure 2.6. The major components of the router are virtual channel buffers, route compute logic, virtual channel allocator, switch allocator and crossbar switch. Buffers holds the flit when it enters the routers. Route compute logic would be compute the next stage router port the flit should traverse. The allocators determines the flits to be selected and sent across the crossbar switch. The crossbar switch is responsible for moving the flits physically from buffers to output ports.

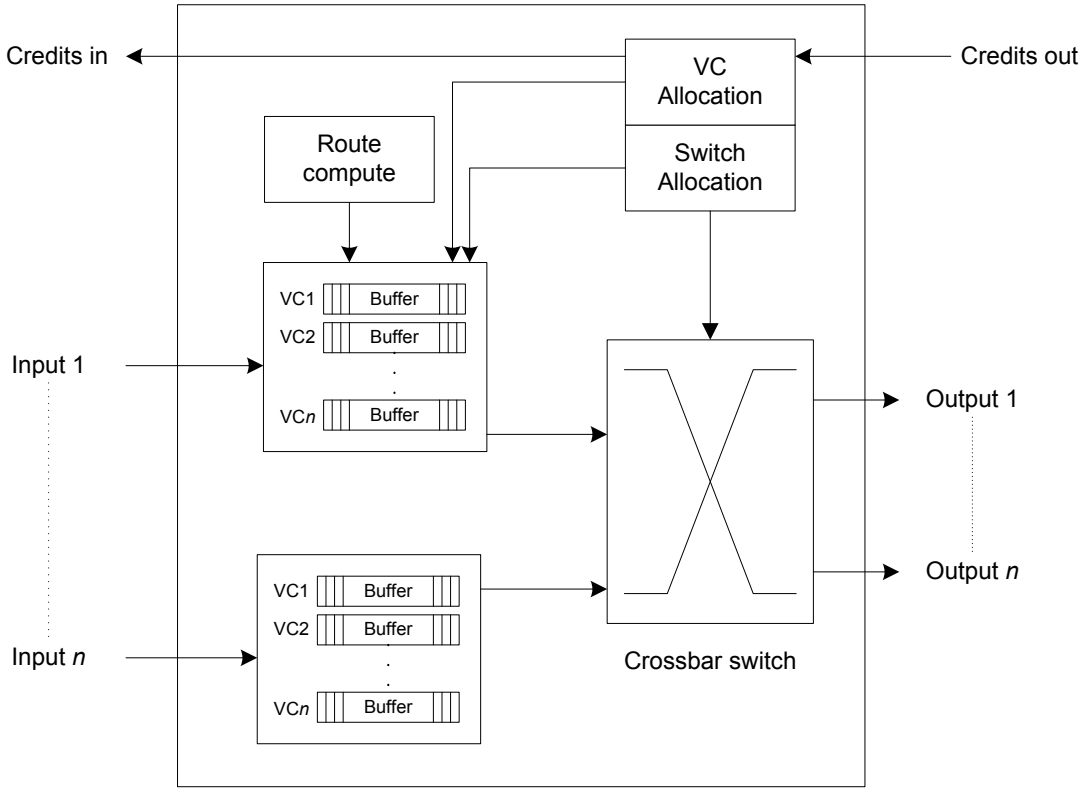



Figure 2.6: Credit based virtual channel router [3]

The organization of buffers has impact on network throughput and sharing of bandwidth by the packets. There are multiple techniques for organization of buffers. Some common techniques used for buffer organization are *a*) Single fixed length queue *b*) Multiple fixed length queue *c*) Multiple variable length queue [3]. Single fixed length queue consists of single queue of fifo buffer with fixed length which can hold one packet at a

time. A multiple fixed length queue also known as virtual channel is set of single fixed length queue fifo buffers. A multiple variable length queue is similar to virtual channels but can accommodate multiple packets simultaneously in single fifo buffer.

In the router we use allocators and arbiters for virtual channel selection, switch allocators. An allocator or arbiters requires to translate more packets succeeding from virtual channel to crossbar switch which would help higher network throughput. Researchers have proposed different types of allocators and arbiters. Some of the commonly used allocators and arbiters are round robin arbiters, matrix arbiters, separable allocator, wavefront allocator etc.

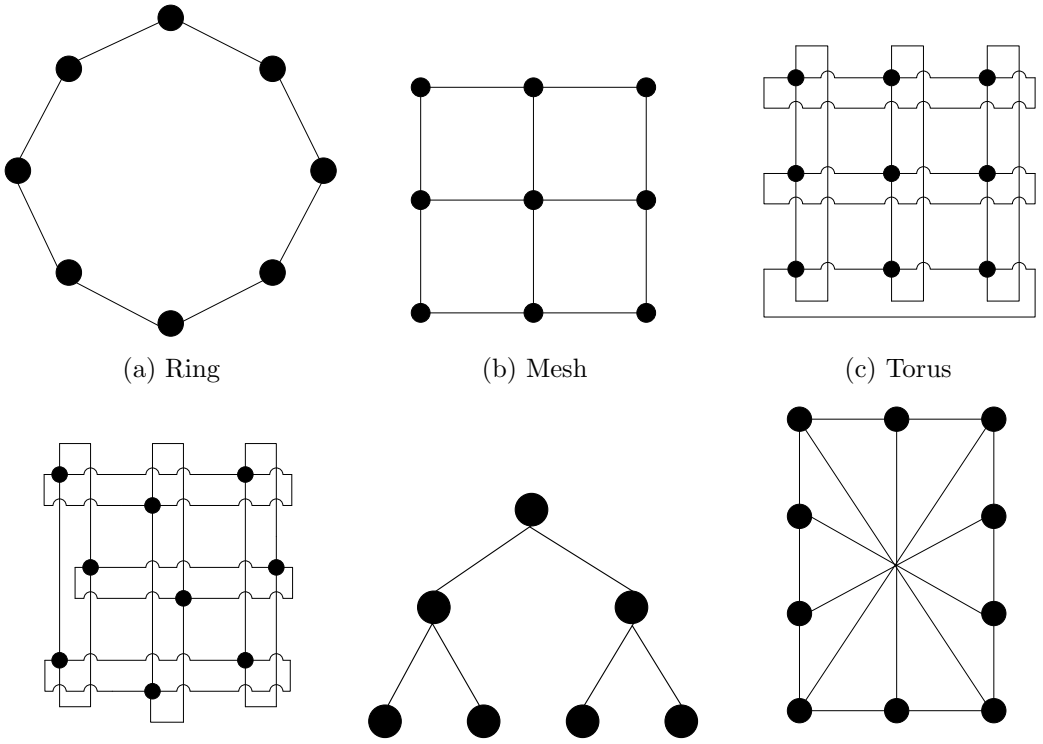
### 2.2.4 Topology

A topology in the NoC determines the physical layout and connections between nodes, routers in the network. It determines the number of routers or hops for the message to traverse from source to destination node. Also it determines the length of the network which influence latency of the NoC. The throughput of the NoC is also significantly determined by topology. Topology provides alternate paths between the nodes, so that traffic can be spread evenly across the NoC and hence reduces network latency and improves network link bandwidth utilization. Complexity of topology implementation is determined by number of links at each router and ease of physical design on a chip (wire lengths and number of metal layers required) [3]. Various topologies like mesh, torus, ring, butterfly, octagon, spidergon and irregular interconnect networks as shown in Figure 2.7 have been analyzed and compared by Dally *et al.* [4] and Pande *et al.* [8].

A mesh architecture is one of the commonly used topologies in the NoC. It can be described as a k-ary-n cubes where k is the number of routers and n is the number of dimensions. Various researchers have analyzed mesh topology and compared with other topologies. The study done by Pande *et al.* [8] shows that the throughput of a mesh for uniform traffic is lower compared to fat tree and octagon topologies. But for localized or neighboring traffic patterns, a mesh provides better performance and throughput. A detailed analysis done by [9], [10], [11] shows spidergon and mesh architecture performance capability is almost similar for similar traffic conditions. But they show that the implementation of a mesh architecture is more complex than a spidergon with increase in area and power. The advantages of mesh as mentioned by [12] shows we can reduce the mesh architecture like concentrated mesh where each router services more than one IP and connecting alternate rows of mesh via express link would provide best results.

A torus topology is similar to mesh topology with edge symmetry. This property helps the torus network to balance traffic across channels. In each router connected in torus topology has same symmetry which is not the case with mesh topology for edge routers. Hence single design of router is enough for complete NoC. A major benefit of torus topology over mesh topology is reduced network diameter which would reduce the maximum number of hops by half and it has larger bisection width.

A ring topology comes under the family of torus topology with k-ary-1 cubes. It is a simple topology due to simplicity in routing algorithm to transport packets which is clockwise or counter clockwise. However scalability is not good compared to other topologies since it does not provide more than 2 paths for a router to spread the traffic



(a) Ring (b) Mesh (c) Torus

(d) Folded torus (e) Fat tree (f) Spidergon

Figure 2.7: NoC topologies

in the network.

A spidergon topology as explained in [11] has an even number of nodes are connected by links to the neighboring nodes in clockwise and counter-clockwise directions plus a cross connection for each pair of nodes. The key characteristics of the spidergon topology include better network diameter and low router degree over mesh topology, homogeneous building blocks (one router design is enough to build the entire network) and simple routing scheme. In [9], Suboh *et al.* present spidergon topology outperforms mesh topology under certain condition because of its regular and stable behaviour at the topology link level. Analysis done by Bononi *et al.* [10] show with a synthetic traffic to create certain hot spots, spidergon topology performs better compare to mesh topology.

Fat tree topology is a k-ary n-tree topology. Study done by Pande *et al.* [8] show fat tree provides performance scalability (>64 cores) but cost of implementation is high (area and power). Floorplanning and physical design in sub micron CMOS technology is complex compared to mesh or spidergon topology. Inter switch wire length is not equal which contributes to area overhead for adding repeaters or buffers (if link transmission constraint is one cycle) [13]. Hence in our thesis we will not focus on fat tree topology for design space exploration.

### 2.2.5 Network interface

Network interfaces are modules which interface nodes with the routers. The key ingredient for design of a network interface is to achieve the decoupling between computation and communication. This would enable the node and interconnect to be designed independently [14]. It will also enable communication and computation to perform independently if required by the application, which results in performance improvement. Network interface functionality can be divided into two parts: *a*) Core part *b*) Network part Responsibility of the network interface is to receive the data from the node, con-

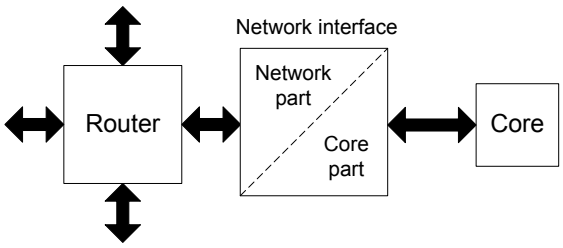



Figure 2.8: Basic network interface

vert it to packets and dispatch into the network. It should also accept packets from the network, retrieve the data from packets and send it to the node. The communication protocol and flow control decision as explained in Section 2.2.2 should be compatible with routing algorithm and topology of the network.

## 2.3 NoC performance evaluation

The performance evaluation of the NoC can be done by three main parameters as explained in [4]

- Cost (area and power)
- Latency
- Throughput

Latency is the time taken for a packet or flit to traverse the network from source to destination. Throughput defines how much data the network can transport from input to output in a fixed amount of time. A high saturation throughput indicates that the network can accept a large amount of traffic before all packets experience high latencies, sustaining higher bandwidth as shown in Figure 2.9. Area required for implementing the circuits on silicon and power consumption translates to cost for the NoC architecture. In this thesis we would not focus on cost function of the NoC since it tightly correlated with circuit level design and technology library used in the SoC. For future work NoCExplorer can be improved to use technology library to estimate cost function of implementing the NoC.

In Figure 2.9 Kumar *et al.* [15] present simulated data for a NoC architecture based on dimension order routing, a large number of virtual channels (8 VC, 24 buffer ports),

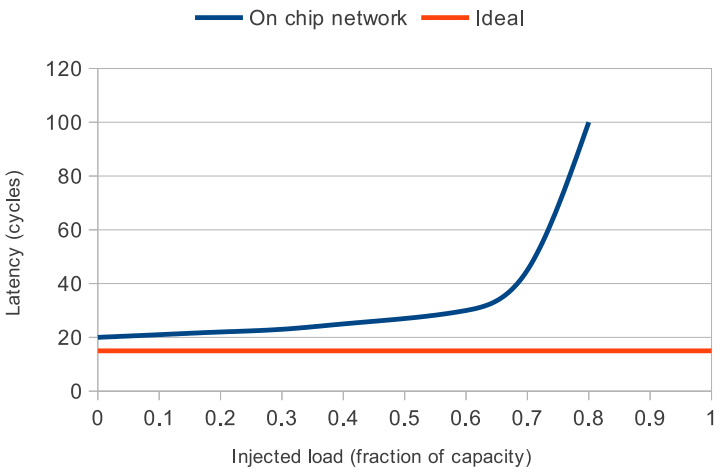



Figure 2.9: Latency and throuput gap for a 7-ary-2 cube [15]

mesh topology (7-ary-2 cube) targetting 65nm, 3GHz and 1.1V. The on-chip network curve shows the gap between ideal latency and throughput. Pande *et al.* [8] show similar trends as shown in Figure 2.9 for torus, butterfly and other topologies under application of synthetic traffic. They show, after the network saturates throughput remains constant even with increase in virtual channels in the router. Even with various combination of NoC architectures as we discussed in previous section, network latency would be higher compared to ideal latency of wire transmission latency. We will use this setup as reference to evaluate NoCExplorer.

## 2.4 NoC simulators

There are several different evaluation tools and methodologies to facilitate research on NoC. Each tool developed tries to cover one or more aspects of NoC design space exploration like

- Configuration of nodes
- Configuration of the NoC like topology, routing algorithms, virtual channels and others.
- Data communication requirements.
- Benchmarking and analysis of results.

Various NoC simulators have been built for evaluation and design space exploration of the NoC. Cristinel Ababei etc [16] and Achballah etc [17] provides a list NoC simulators and tools available to simulate and analyze different types of NoC. In this section we would analyze and study some of the open source NoC simulators. Each simulator has various capabilities but also have few limitations. In NoCExplorer we have included the

capabilities offered by the simulators and addressed a set of limitations, which would be explained in detail in Chapter 3.

### Booksim

Booksim [18] is a open source simulator developed using C++. It is a simulator for network interconnect which supports virtual channel (VC) routers. It supports 10 topologies like mesh, torus, cmesh, fat tree and others. Twenty routing algorithms can be configured to direct packets for the supported topologies. Links between the routers and nodes is set to one, hence multiple links cannot be configured. Packet injection into the network can be configured. Allocators and virtual channels configuration are parametrized based on configuration file. The router design is based on event driven hence it is not cycle accurate. Area, power, hot spot analysis cannot be performed in Booksim. It support various synthetic traffic generator but we cannot apply custom traffic over the NoC. Support for mixed language simulation is not available.

### Noxim

Noxim [19] is a NoC simulator developed using SystemC. It has command line interface to parametrize various components of NoC. In Noxim we can customize network size, buffer size, packet size, routing algorithm, injection rate and traffic pattern. Noxim only supports mesh topology with wormhole routers over synthetic traffic patterns. Support for mapping custom application on the NoC is not available. Evaluation of NoC is done by producing results in terms of throughput, delay and power consumption. Detailed evaluation metrics can be analyzed like total number packets or flits sent or received, global average throughput, maximum and minimum global delay, total energy consumption etc. Similar to Booksim it does not support area, power, hot spot analysis and mixed language simulation.

### NoCTweak

NoCTweak [20] is similar to Noxim simulator developed for cycle accuracy simulation of NoC. The simulator has been developed using SystemC. It supports only 2D mesh topology with each node consisting of core and network interface. For traffic generation is has support for synthetic traffic and embedded application traces with capability to map the application on each node. Router settings are parametrized with various options on virtual channel, buffer depth, routing algorithm, switch arbitration etc. It has support to read power models for early assessment of power consumption in the NoC. It generates statistic outputs like network latency, network throughput, power consumption etc for evaluation of NoC.

### NoCBench and NoCSim

NoCBench [21] is a benchmarking platform to evaluate NoC. The core engine of the simulator is based on NoCSim NoC simulator. It provides an integrated simulation environment with set of standard NoC components and cores. Its main feature is to

able integrated different embedded cores and network components. NoCSim simulator is developed using SystemC. The NoC has to be modeled using manual and XML based configuration. The simulator would connect components from set of core library, simulate and generate reports.

### Atlas

Atlas [22] is a framework which automates various process related to design flow of NoC. The NoC components are described in VHDL and testbench has been developed using SystemC. The tool can be configured to parametrize network dimension, communication channel width, buffer depth, flow control, virtual channel selection and routing algorithms. It supports only synthetic traffic application and mapping on NoC. It has predefined power models for each component and can provide early power estimation of the NoC.

# 3 NoCExplorer design

Execution of single or multiple applications on a Multi Processor System on Chip (MPSoC) is hard to predict due to dynamics of the applications and complexity of hardware. Software designers would need an early predictable communication model of the MPSoC. Hardware designers would need an early estimation of latency, throughput and cost of implementing a Network on Chip (NoC). NoCExplorer is a set of tools in the framework, which can be used by the designers to do design space exploration of NoCs. Based on an input configuration file it can configure and simulate the NoC using SystemC modules. Python based modules provide utility to retrieve, interpret and present the performance of the NoC in a flexible, user customizable way for the designers. In this chapter we describe the complete framework of NoCExplorer. We explain in detail about the SystemC and python based modules and its interaction in NoCExplorer. Usage of NoCExplorer is shown with an example NoC architecture. Value addition of NoCExplorer over other open source simulators is explained at end of the chapter.

## 3.1 NoCExplorer specification

In the previous chapters we have discussed about the NoC components and performance evaluation of the NoC. Based on literature survey of the existing simulators and design space exploration of the NoC architectures, the NoCExplorer specification is derived. The main specifications of NoCExplorer are

**Configuration and simulation:** NoCExplorer is able to configure and simulate behavior of a NoC. To achieve that it should be able to support multiple configurations like

- **Topology:** Support for mesh, torus, folded torus and spidergon topology based on literature survey as discussed in Chapter 2. NoCExplorer is able to simulate other topologies as well if designers add required modules.
- **Routing algorithm:** Similar to topologies NoCExplorer supports multiple routing algorithms like XY routing, turn based model for mesh topology. Torus XY routing algorithm is enabled for tours topology. Routing across first or last is supported for spidergon topology.
- **Network size:** For mesh based topologies (mesh, torus and folded torus), the number of routers in the NoC can be configured in both X and Y direction. Number of node can be configured for spidergon topology.
- **Physical links:** Links between routers can be configured which we call it as router physical links. Network interface links to routers is called master network

interface link and slave network interface link, which is also configurable to have one or more number of links.

**Virtual channels:** Virtual channels can be modeled based on number of channels, buffer depth, virtual channel allocator and virtual channel arbiter.

**Clock:** NoC and node can be configured for different operating frequency.

**Mixed language simulation:** NoC configuration, generation and mapping of application is based on SystemC implementation. SystemC is more suitable for higher level modeling like transaction level modeling with capability to do system level modeling and software development. Also simulators like Questasim from Mentor Graphics support mixed language simulation with SystemC and VHDL.

**Traffic generator:** NoCExplorer is able to generate traffic for the NoC like

**Synthetic / Custom traffic:** Traffic modeling supports configuration of maximum and average bandwidth, burst parameters, destination, number of packets for simulation. Packet injection rate is derived by NoCExplorer based on bandwidth and burst parameters, which is explained in this chapter.

**Flit interval selection:** Detail configuration support to model flit interval rate into the network.

**Simulation time parameters:** Each node start and stop time can be configured.

**Results:** NoCExplorer can generate min, max and average packet latency, flit latency of the network and others. Link utilization and router congestion can be visually represented using heat maps. Each flit is tracked and captured in the results, hence missing flits can be highlighted.

## 3.2 NoCExplorer framework

Framework of NoCExplorer is shown in Figure 3.1 consists of

- Traffic modeling
- NoC configuration
- NoCExplorer
- Simulation and analysis of results

Designers can apply application transaction model onto the NoC. They can also use in built-in synthetic traffic generator for design space exploration when actual application traffic is not available. Based on traffic parameters, designers can choose and combine various parameter of NoC like topology, routing algorithm, virtual channel buffer depth and others to derive desired NoC architecture. NoCExplorer as shown in Figure 3.2 would read the configuration parameters set by the designers. It would connect the NoC architecture from NoC building blocks present in the framework and configure nodes to generate traffic based on application requirements.

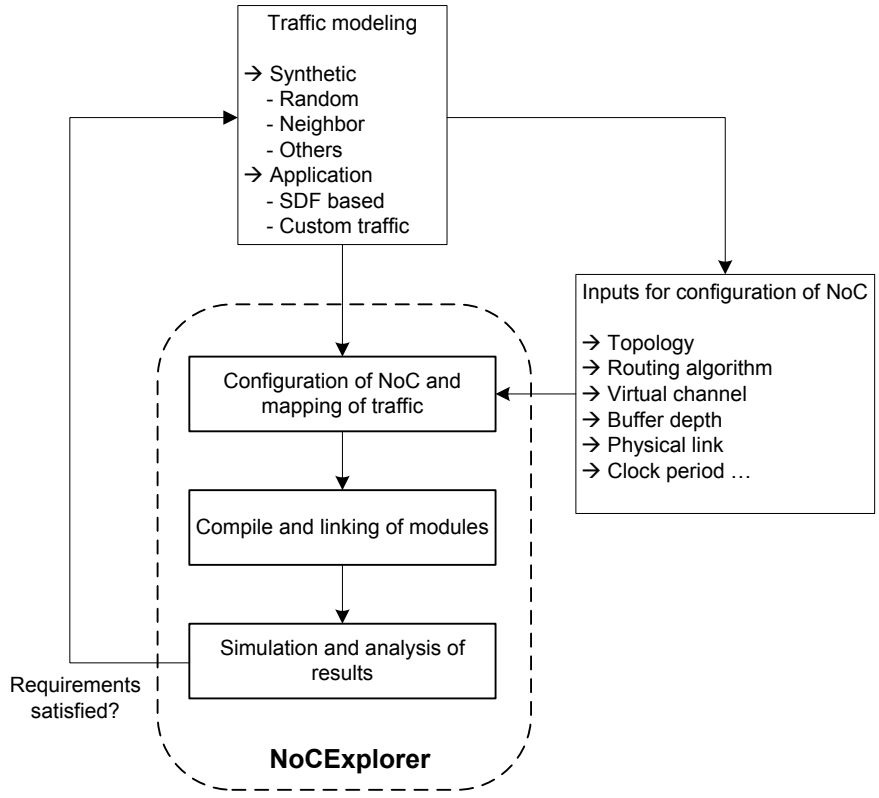



Figure 3.1: NoCExplorer framework

NoCExplorer modules are divided into SystemC and Python based implementation as shown in Figure 3.2. SystemC modules are compiled and it simulates the network for given constraints. Results generated from SystemC modules are retrieved and interpreted and presented using python based modules. Designers can use these results to see if application requirements are satisfied. If it is not satisfied, designers can redo the process with a different NoC configuration.

SystemC modules of NoCExplorer is divided into various blocks in terms of hardware functionality and to handle the traffic. Each block interface ports are modeled in terms of register transfer level signals. This provides capability to swap the module with VHDL or Verilog RTL blocks for mixed language simulation and enable more accurate assessment of the NoC. NoCExplorer SystemC modules as shown in Figure 3.3 is divided into main three modules *a*) NoC library *b*) Traffic generator *c*) Traffic manager. NoC library module contains the main hardware blocks to configure NoC architecture like router and network interface. Topology module connects the router, network interfaces and nodes to create NoC entity. Network interfaces consists of master and slave network interface. Router module has various sub modules like crossbar, virtual channel, route compute and others which can be considered as heart of the NoC. Each sub block under these blocks is explained in detail in following sections of the chapter. Traffic generator module consists of multiple node modules which is responsible to generate traffic based on user configuration. It can be custom traffic or synthetic traffic. Traffic manager is

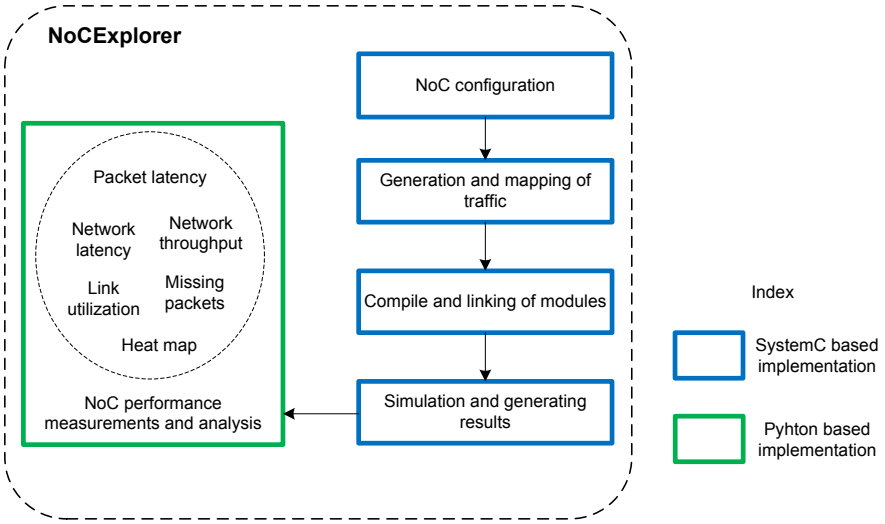



Figure 3.2: NoCExplorer block diagram

like an observer which keeps track of communication requests between routers, network interfaces and nodes. It monitors entry and exits of flits in the network. This information is used to write output files which will be used later for interpretation and analysis by Python based modules.

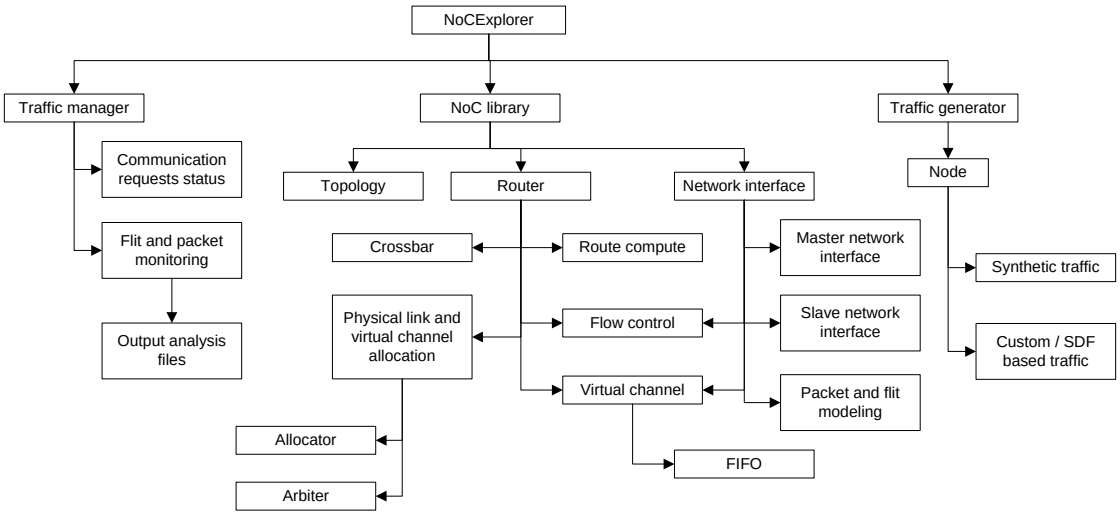



Figure 3.3: Hierarchy of SystemC modules in NoCExplorer

Integrating of modules in NoCExplorer is required to make the data flow and control signal exchange possible as shown in Figure 3.4. The data is generated from the nodes which are interfaced with network interfaces. The master network interface assembles the flits and packets. It is stored in buffers, ready to be injected into the network. Routers accept the flits and compute the next direction of the flits. It is once again stored in buffers till next stage router or slave network interface is ready to accept the flit. The

flits traverse through crossbar to reach next stage router or slave network interface. Once flit reaches slave network interface it is stored in buffer and when node is ready to accept the flits it is transferred. Node consumes the flit data if required and the data is also sent to traffic manager. Traffic manager process the flit to extract information required for profiling and output files are written for post simulation analysis. Control signals are exchanged for arbitration and allocation of resources in the block.

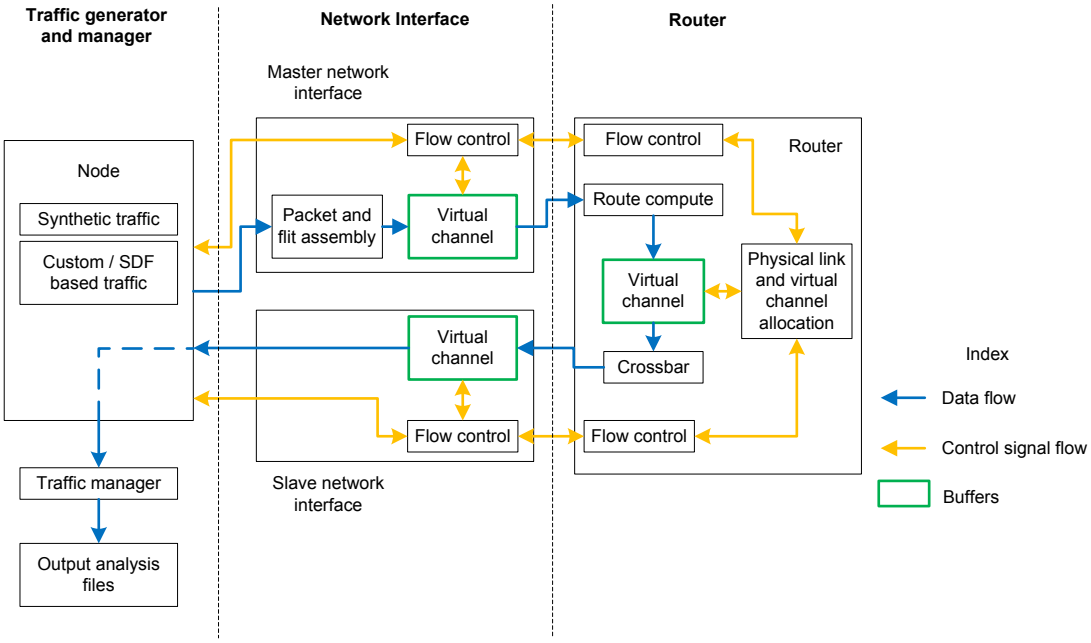



Figure 3.4: Data and control signal flow in NoCExplorer

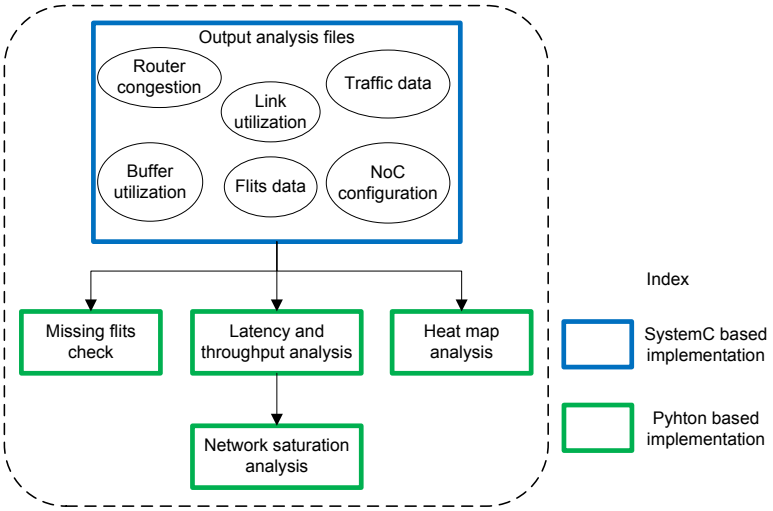



Figure 3.5: Python based modules of NoCExplorer

Python modules of NoCExplorer as shown in Figure 3.5 main function is to retrieve, interpret and present the performance of the NoC in a flexible, user customizable way for the designers. The inputs for the utility scripts are generated by SystemC modules of NoCExplorer. These modules can be executed after the simulation of SystemC modules is completed. It is based on csv file format with flit, packet, router congestion, link utilization and others information encoded in it. Various utility scripts use these inputs files to check missing flits in the NoC, analyze latency and throughput of the NoC, generating heat map and to find network saturation point.

## 3.3 NoCExplorer library

### 3.3.1 Packet and flit assembly

In a NoC data transmission from source to destination is based on encoding information in forms of packets. Packets are set of flits combined together which was explained in previous chapter. Flits are classified as head flit, body flit and tail flit. In NoCExplorer we have modeled flits to carry data and control signals required for routers and network interface. We have used combination of C++ and SystemC data types to model a flit as shown in Appendix B to reduce simulation run time. If the data width is configured as 32 bits, to transport 24 bytes of data a packet size of 6 flits is required. It will contain 4 body flits (each carrying 8 bytes of data), 1 head flit and 1 tail flit.

Each flit consists of following information relevant for communication

**Output port direction:** Contains direction for the packet to proceed in the network.

**Flit type:** To represent head, body or tail flit.

**Flit number:** Unique number is assigned to each flit, so that when packet is disassembled data sequence can be restored.

**Flit data:** Data being transported in the network.

**Virtual channel number:** Virtual channel to use in the router.

We also include more information in each flit which is used for profiling the performance of the NoC post simulation. This information is not included in actual implementation of the NoC. They are

**Source and destination node:** Source and destination node numbers.

**Packet ID:** Unique packet ID which helps in monitoring packet movement in the network.

**Timestamp of flits:** It is used to measure latency of the flits.

**Hop count:** Used to measure the performance of routing algorithms.

**Physical link numbers:** Used to measure link utilization in the network.

### 3.3.2 Network interface

The network interface design for a node and a router is split into two functional blocks. The master network interface accepts requests from the node, generates and pushes flits into the network. The slave network interface receives flits from the router and sends data to the node. Our network interface design has support for multiple physical links which can be parametrized in the user configuration file. Network interface consists of virtual channels to store flits similar to the operation explained in Section 3.3.3.

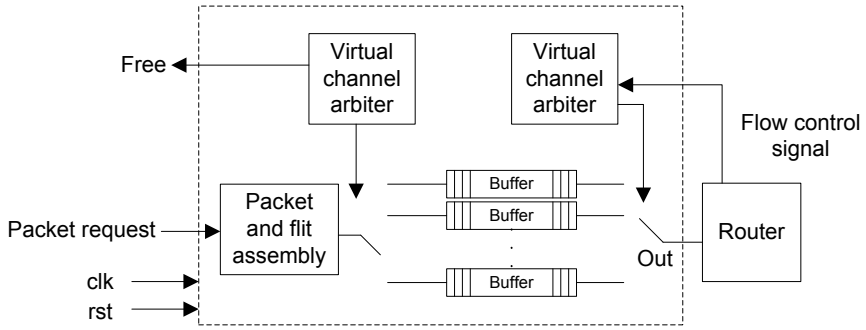



Figure 3.6: Master network interface block diagram

The master network interface as shown in Figure 3.6 has been designed to accept the request from the node to transfer data. The node provides following information to the master network interface *a*) Size of the data *b*) Unique packet id *c*) Source node *d*) Destination node *e*) Allotted time for packet *f*) Flit rate selection mode. When master network interface receives a request from the node, it assembles the packets and flits as explained in Section 3.3.1. Once flit is assembled it is stored in the buffer of a virtual channel. Designers can configure allocation scheme for virtual channels. We support priority and round robin allocation to fill up virtual channels. Master network interface signals the node if its busy in operation or free to accept next packet request. We do not send credits to the nodes since it does not use the information. The flow control with router is based on credit based handshake between nodes and routers as explained in Section 2.2.2. Credit is decreased as flits fill up the buffers in the virtual channels. After flits enter the virtual channel the master network interface checks if full credits in a buffer is available in next stage router. It is because in our router design we do not allow two packets to be stored in a buffer, to avoid extra control signals in virtual channel allocator. Flits are transmitted via physical links if required credits are available in next stage router. If physical link resources is not available or zero credits in next stage router due to buffer backpressure the flits waits in the virtual channel.

Based on flit interval selection mode, flits are written in virtual channel buffers at certain time intervals. Interval between flits has impacts the performance of the network which will be explained in Chapter 4. In our design we support different flit interval selection method as shown in Figure 3.7. Fixed flit interval has equal delay between flits when packet is written into virtual channel. One cycle flit interval generates and writes one flit per cycle. This is possible when complete data is available for transmission through the NoC. Random flit interval is modeled to provide random delay between

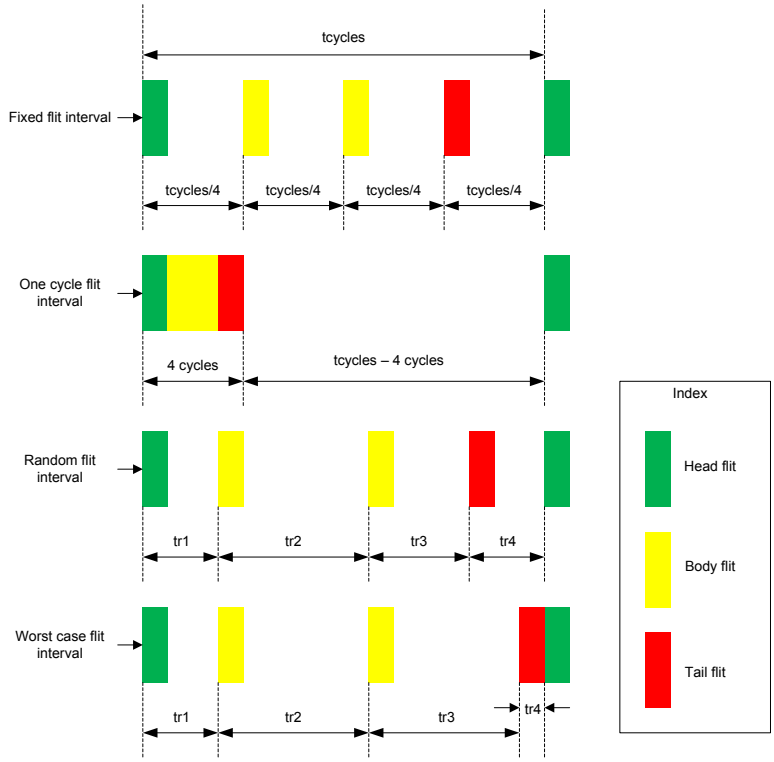



Figure 3.7: Flit interval selection method

flits. Worst case flit interval will equally divide intervals between flits and finish sending complete packet data for the time interval requested by the node. If there is buffer backpressure, flits injection is forwarded by one cycle till resources are available in next stage. If this exceeds packet interval time, flits are injected with one cycle interval.

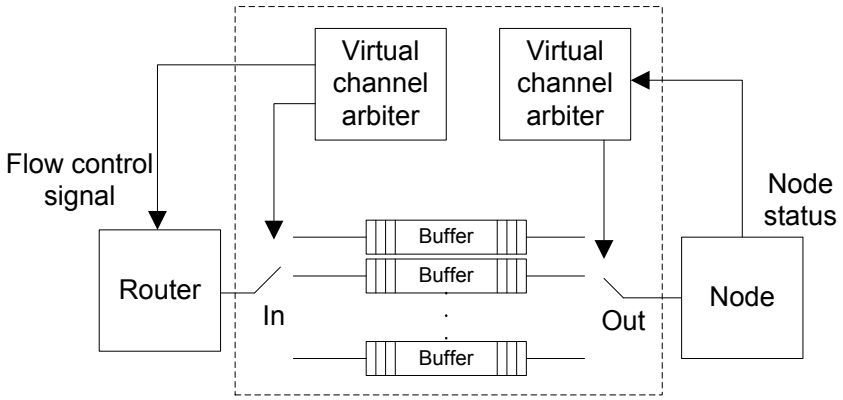



Figure 3.8: Slave network interface block diagram

The slave network interface as shown in Figure 3.8 has been designed to accept the flit data from routers. It is stored in virtual channel buffers and sent to the node if it is free. Current design does not include disassembling of packets into bare data, since flit

level analysis is done in detail at traffic manager. Before sending the flits to the node, it is timestamped to mark exit of flits from the network. This information is used in profiling to measure latency of the flits and packet.

### 3.3.3 Router

Router block is an integration of route compute, virtual channels, physical link and virtual channel allocator, crossbar and registers as shown in Figure 3.9. Based on number of router input or output ports which is determined by topology selection, each block is required to be instantiated multiple times. For example in mesh topology number of ports required for router would be 5 (1 network interface + 4 router interface). Hence route compute and virtual channel has to be instantiated five times. Physical link and virtual channel allocator and crossbar is instantiated once to handle parametrized router ports.

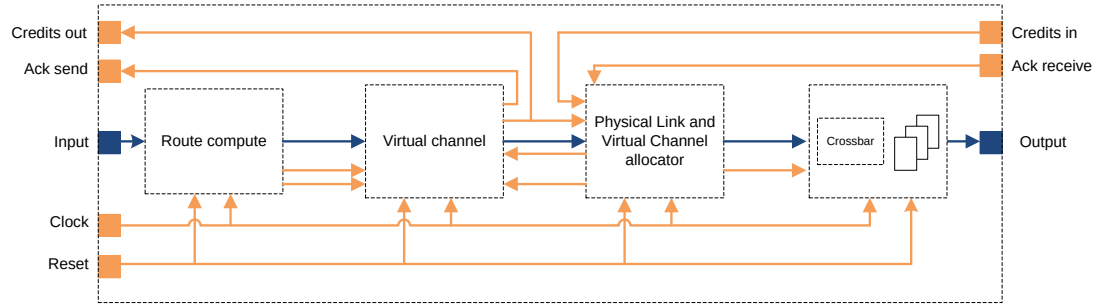



Figure 3.9: Block diagram of router block

In the router each block delay parameter can be modeled to mimic pipeline requirements. For our experiments we have considered 2 cycle delay in routers. In one clock cycle, flit is entered, route compute and write operation in virtual channel is completed. In next clock cycle flit can be read from virtual channel and written to output registers via crossbar datapath. User can configure the cycle delay in routers based on hardware requirements. Designers can use this feature to create a new NoC architecture and to assess the latency impact of a router.

The flow control signals connected to the router is based on credit based handshake between network interfaces and routers as explained in Section 2.2.2. Credits are sent to previous stage when flits enter the router. Credits are received when flits exit the router. So when tail flits exits the router the credits are not sent to previous stage till next stage acknowledges it has received the tail flit. It is to make sure flits are not dropped in the data path if next stage does not accept the flit due to buffer backpressure. Registers are placed at the output of crossbar to pipeline the data path.

#### Route compute

Route compute block determines to which of the neighboring routers or network interface the packets needs to be sent. In NoCExplorer we propose route compute block to be the first stage in the router. It reads the incoming flits from neighboring routers or network

interface. Since router architecture is based on wormhole routing, route computation is done only for head flits. Rest of the flits will follow the same direction as computed for head flit.

When head flit enters the router it is time stamped and saved in flit to capture arrival time of the flit. It is used later for analysis and computation of router congestion and link utilization. Based on the user defined parameter corresponding routing algorithm routine is executed to find the output port direction. Each routing algorithm is defined as a separate routine since it is modular and easier to add new routing algorithms. We have implemented following routing algorithm to support various topologies

- XY routing
- Routing west first
- Routing south last
- XY routing torus
- Routing across first
- Routing across last

Pseudo code for each algorithm is mentioned in Appendix A.

### Virtual channel

Virtual channel is a set of FIFO buffers which is parametrized for number of buffers and buffer depth. FIFO buffers are modeled to store flits. In our design we accept one input and multiple output ports based on number of buffers. The first flit to exit the buffer can be read by allocators and arbiters. Write select signal is used to select the buffer number for flit write operation. Since our design is based on wormhole router architecture, virtual channel has a logical block to provide acknowledgement when tail flit is received. This is used for previous stage router or network interface to free the resources allocated for the packet.

### Crossbar

Crossbar design is the heart of datapath in router design. The crossbar module is responsible for moving the flits physically from buffers to output ports. In our design we support parametrized switch input and output ports. Each input port would have select signal to write flits on output port, which is controlled physical link allocator. It supports reset signal to write empty flits on output port.

### Physical link and virtual channel allocation

Physical and virtual channel allocation block is designed to handle various operations. It reads flit from the output ports of virtual channel. Sequence of flit read operation is based on arbitration scheme selected by the user configuration. The current design supports round robin arbitration and priority based arbitration. If the flit is head flit,

route computation data is read from flit. Next stage virtual channel selection is based on user configuration. The current design supports network interface based selection and dynamic selection. In network interface based selection virtual channel number is kept constant from source to destination. In dynamic selection each next stage virtual channel status is read till empty buffer is found.

Body and tail flit follow the head flit (based on wormhole router architecture), hence arbitration for virtual channel is avoided for these flits. Once tail flit is written at output port it waits for acknowledgement from next stage router or network interface before the resources are freed to be used for other packets.

### 3.3.4 Topology

Topology is created by linking routers, network interface and nodes based on user configuration. In our design we support mesh, torus, folded torus and spidergon topology. Based number of nodes to be connected, size of the network is determined for the topologies. Each node is interfaced via master network interface and slave network interface to the router. Each router, master network interface and slave network interface control and data signals are connected form the topology. In this process each block is given unique node ID to identify source and destination node for traffic generators and route computation.

## 3.4 Node

NoCExplorer provides rich set of controls to model a node to inject and accept a packet into the network. A node can be designed and connected to the network by the designers. Currently each network interfaces can be connected to a node. Designers can model detail operation of the node but it is not mandatory.

NoCExplorer supports modeling of a node based on synthetic traffic or custom traffic. To achieve it we support a set of parameters for each physical link in the node. This allows multiple applications can be modeled for each physical link in a node. The traffic in the node can be modeled as follows

**Destination node selection:** NoCExplorer support automatic selection of destination nodes which is used in application of synthetic traffic or user provided destination node. There is support for random, fixed, neighboring and others destination node selection.

**Data size:** Designers can specify amount of data to be sent in each packet. For example if data width in NoCExplorer configuration is set to 32 bits, then each body flit can send 4 bytes of data. Hence packet size is directly proportional to size of the data.

**Operation period:** Each node start and end of its operation can be modeled. Start time can be set for a node to mimic warm up time of a system. A node operation can be stopped by specifying end time. It can also be stopped after sending certain amount of data or certain number of packets into the network.

**Bandwidth:** Bandwidth of the node determines the rate of data transfer into the network. NoCExplorer translates the bandwidth to determine the injection of flits and packet into the network in terms of flits/cycle.

**Internal memory:** Memory or buffer modeling in the node determines amount of data it can accept before it can start traffic operation. Once required amount of data is received by a node it can trigger the process to start data transfer. This is used extensively for modeling of custom applications.

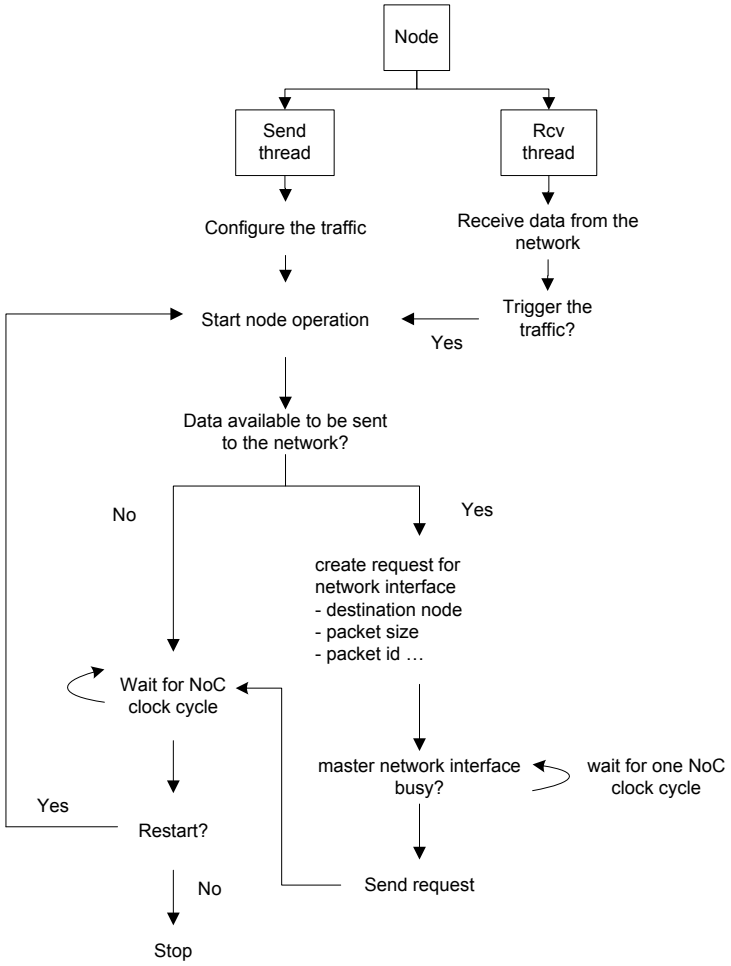



Figure 3.10: Node modeling flow chart

To generate the traffic, NoCExplorer divides the operation to send and receive the data into the network in two separate threads as shown in Figure 3.10. Receive thread receives the data from the network and it can be modeled to trigger operation in send thread. Send thread configures the traffic with parameters as explained in this chapter. When data is available, request for data transfer is sent to master network interface. Based on operation period set by designers the send thread is stopped or restarted.

### 3.4.1 Synthetic traffic

Synthetic traffic are mathematical based traffic patterns mimicking spatial and temporal distribution of data transfers while abstracting the program functionality. In NoCExplorer synthetic traffic can be applied using node modeling configuration parameters. Destination node selection be set to random, fixed, neighboring and others mode. For example a synthetic traffic can be applied on a node as shown in Figure 3.11. In the example various parameters can be set on a node, which are

**Time:** The node starts operation after warmup time of 1sec and ends after 5.5 secs.

**Bandwidth:** Node requires average bandwidth of 100MB/sec.

**Burst size:** Data size sent into the network can be between values max and min burst size parameter. In this example min burst size is 4KB and max burst size is 200MB. Burst size value is always bounded by average and max bandwidth allowed by the designer.

**Packet interval:** NoCExplorer derives the max and min packet interval time from bandwidth parameter set by the designer.

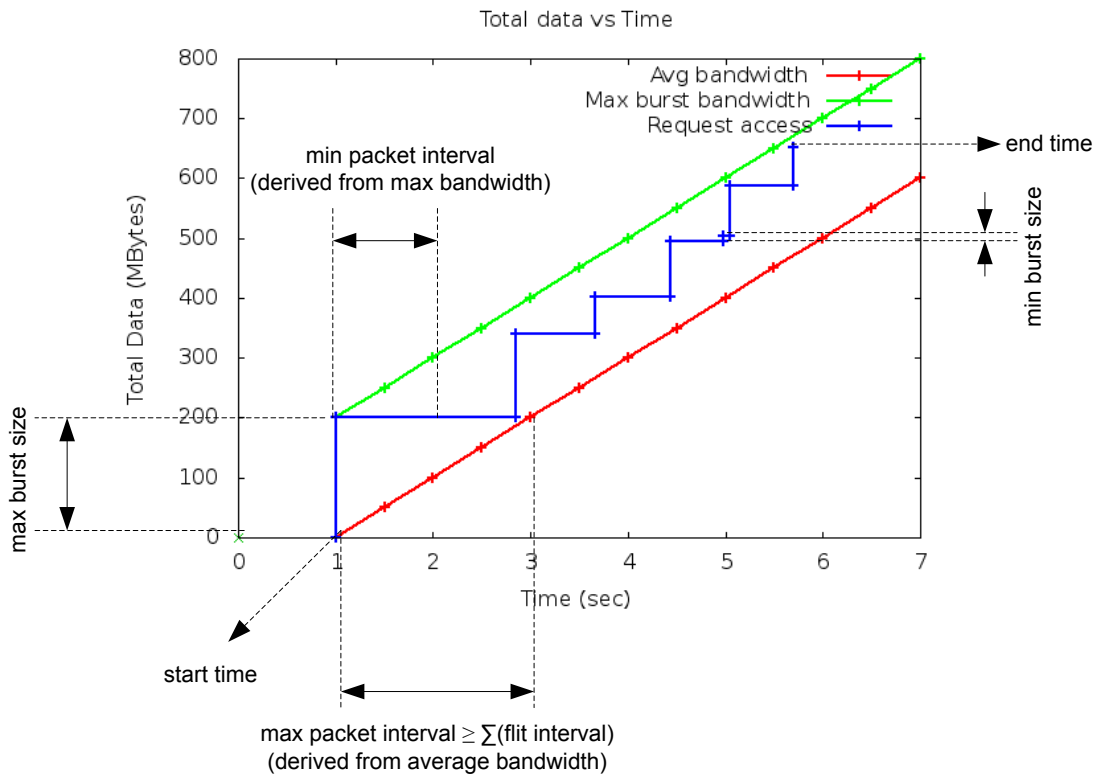



Figure 3.11: Data transfer request from a node

### 3.4.2 Custom traffic

By using parameterizable knobs in node modeling custom application based traffic can be applied on NoC. Researchers have proposed various methods in modeling of an application. In NoCExplorer we will use Synchronous Data Flow (SDF) [23] based application model to apply custom traffic on NoC. For more details on SDF, Schaumont *et al* [24] can be referred with examples on embedded video applications.

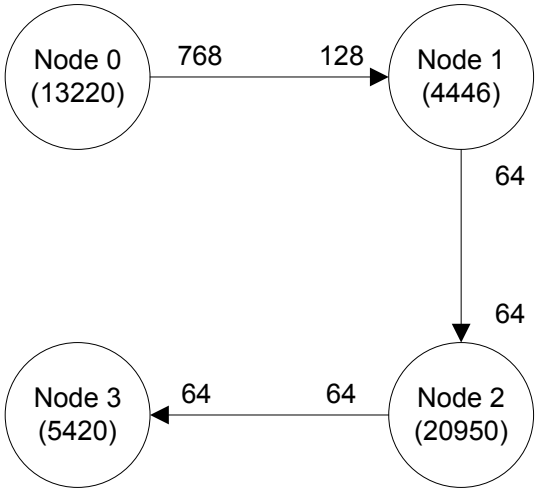



Figure 3.12: Example SDF graph

An example SDF graph is shown in Figure 3.12 which can be mapped onto a NoC using NoCExplorer. In this example, we have assumed each task is mapped a node. Designers can also model multiple tasks in a node. Each node takes certain amount of cycles before output data are generated. For few nodes certain amount of data is required to trigger the process. Using these parameters bandwidth, burst size and other parameters required to model a node can be calculated as shown in Equation 3.1 and Table 3.1.

$$
\begin{aligned}
&Clock\ period = 2ns\ (assumption)\\
&Data\ per\ operation for node 0 = 768\ Bytes\ (assuming\ 1\ token = 1\ Byte)\\
&Cycle\ count\ per\ operation\ for\ node\ 0 = 13220\ cycles (assuming\ unit\ of\ time\ is\ cycles)\\
&Total\ cycles\ for\ one\ operation\ 768\ tokens\ (from\ Node\ 0\ to\ Node\ 3)\\
&= Node\ 0 + Node\ 1 + (Node\ 2 * 6) + Node\ 3 = 148786\ cycles\\
&Max\ bandwidth = Data/(Cycle * Clock\ period)\\
&= 768/(13220 * 2ns) = 29.1 MBytes/sec\\
&Average\ bandwidth = 768/(148786 * 2ns) = 5.16\ MBytes/Sec
\end{aligned}
\tag{3.1}
$$

NoCExplorer also has capabilities for designers to send actual data/content from source to destination. Designers can implement a real application that does actual computation on actual data and delivers actual output to the network. In our experiments

Table 3.1: Node modeling for a SDF graph

| Option | Node 0 | Node 1 | Node 2 |
|---|---|---|---|
| dst node | 1 | 2 | 3 |
| dst node selection | Fixed dst node | Fixed dst node | Fixed dst node |
| data size | 768 bytes | 64 bytes | 64 bytes |
| start time | 100ps | 0ps | 0ps |
| average bw | 2.90E+7 | 7.20E+6 | 1.53E+6 |
| max burst size | 768 bytes | 64 bytes | 64 bytes |
| min burst size | 28 bytes | 28 bytes | 28 bytes |
| flit interval selection | One cycle | One cycle | One cycle |
| input method | Source | Intermediate | Intermediate |
| int mem | 1024 bytes | 1024 bytes | 1024 bytes |
| req data | 0 | 128 bytes | 64 bytes |

we have disabled this feature since we did not require real data and high simulation run time. In Chapter 4 we have explained in detail about simulation run time.

## 3.5 Traffic manager

Traffic manager main operation in NoCExplorer is to monitor the flow of flits and write output files for post simulation analysis. Single block of traffic manager is connected to multiple slave network interfaces to read the flits from the network. As explained in Section 3.3.1 each flit contains required information which is processed by the block. The operations of the block are

**Time stamp:** Each flit is time stamped after it exits the network to measure latency.

**Flit sequence:** Traffic manager keeps track of flit order in a packet when it is sent from the slave network. When out of sequence flit is detected, immediate SystemC error is reported and simulation is stopped. This error indicates the given NoC architecture has flaw in its design. To resolve the issue NoCExplorer provides debug flags (constant.h) for advanced users to print flit information when it travels through the NoC.

**Output files:** Multiple output files are written with flit and packet information for post simulation analysis.

All the output files generated by traffic manager is based on csv file format. Details of each output files as follows

**outputFlit.csv:** This file contains detailed information of each flit processed in the NoC. It contains packet id, flit sequence number, in and out time of the flit, source and destination node, hop count and others.

**trafficPattern.csv:** Traffic pattern of each node is captured in this file. Each packet request information which includes packet id, source and destination node, allo-

cated time for the packet, start time of the packet and others. This is used as reference by python based modules to locate missing packets in the NoC.

**nocConfig.csv:** NoC configuration parameters set by the designer like topology, routing algorithm, buffer depth, clock frequency and others is written in nocConfig.csv file. It is a reference file for back trace the settings used in NoCExplorer.

**bufferUtil.csv:** This file captures buffer utilization in each router and network interfaces for complete simulation time. In our experiments we found file size was more than 10GB since cycle level information was being written. Hence for default settings this feature is disabled in NoCExplorer. For future work the data written in the file can be optimized for low file size.

**routerCongestion.csv:** Traffic manager calculates performance of each router and network interfaces to perform operations per flit. When resources are available for routers and network interfaces, it should be able to process one flit per cycle. If there is congestion in the NoC, routers and network interfaces can take more than 1 cycle / flit. The output file captures average, min and max number of cycles / flit operation in each router and network interface.

**linkUtilization.csv:** Similar to router congestion, link utilization output file captures utilization of links in the NoC for complete simulation time. From this data we can fine unused links in the NoC. The data is represented in the form of percentage. For example, 100% utilization in a link conveys that a flit is transported in each cycle.

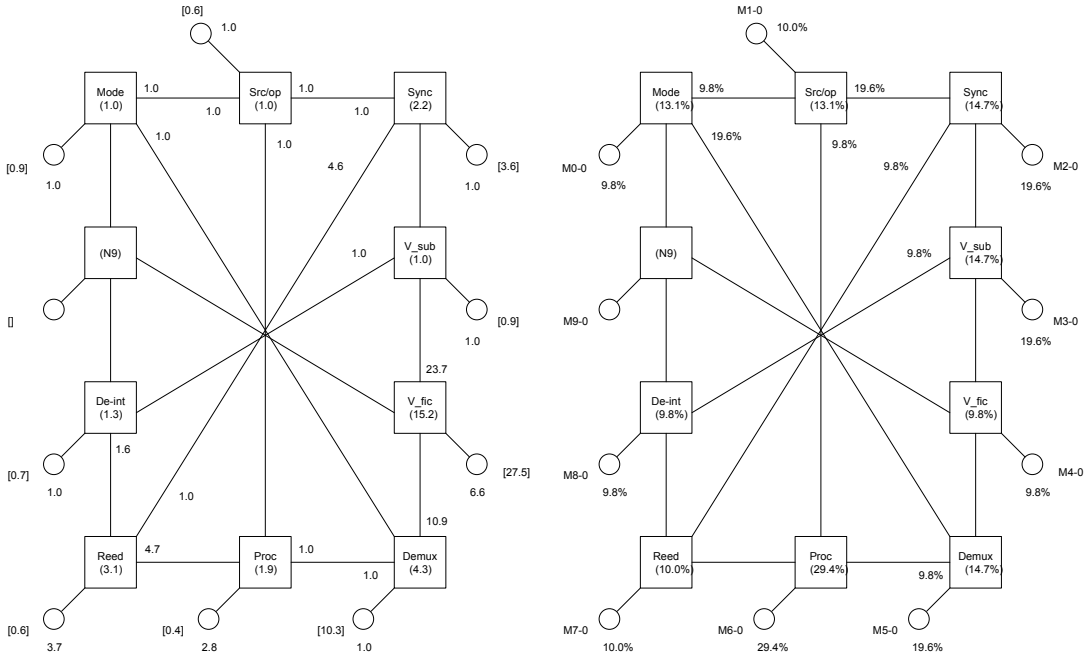
## 3.6 Python based utility tools

NoCExplorer provides multiple utility tools based on python for post simulation analysis to evaluate the performance of the NoC. They used output files generated by SystemC modules as explained in Section 3.5. Details of each utility tool as follows

**Missing flits:** Using output flit and traffic pattern data we can check if there is any flit or packet still in NoC even after simulation window is closed. If active flits are still in the network, it could be because simulation window time is not enough or due to NoC architecture issues. Deadlocks in the network due to certain routing algorithms can be easily debugged using this utility tool. It generates a output file with missing packet information like packet id, source and destination node, start time of each flit and others.

**Latency and throughput analysis:** This utility tools consumes output flit and traffic pattern data generated by simulation and generate NoC performance results. It calculates average/min/max accepted and ejected load (flits/cycle), virtual channel buffer utilization, average/min/max packet and flit latency. It also combines packets based on hop count and report its average/min/max packet and flit latency. An example output file is shown in Appendix C. It also has capability to merge multiple simulation run for varying injection load to plot network saturation point. We have used this tool extensively and results are shown in Section 4.1.

**Heat map:** Heat map utility scripts generate visual representation of router congestion and links utilization in the topology. Router port congestion as shown in Figure 3.13a shows for each port how many cycles are required to transfer one flit (cycle/flit). Link utilization as shown in Figure 3.13b shows load factor in each physical link. In the Figure 3.13 numbers shown with '[]' represent master network interface. Numbers represented next to the router output ports connected to the network. Average value of router congestion or link utilization is shown inside the router block.




(a) Router port congestion map (b) Link utilization map

Figure 3.13: Example heat map of NoC

## 3.7 NoC simulator comparison

We studied various simulators in terms of parameterizable library components, analysis methods, simulation platforms and hardware integration as explained in Section 2.4. Observation from the analysis of other simulators helped us to do better implementation of NoCExplorer. In Table 3.2 we have captured and compared our framework with other simulators.

Major advantages of NoCExplorer is capability to parametrize network size, physical links and others. Detailed buffer modeling can be done to set buffer depth and number of buffers in virtual channel. With parameters to control injection of packets into the network, NoCExplorer has advantage over other simulators to control even flit injection rate into the network. This can used by the designers to model a node when the ap-

Table 3.2: NoC simulator characteristics

| **Parameters** | **Booksim** | **Noxim** | **NoCTweak** | **NoCBench** | **Atlas** | **NoCExplorer** |
|---|---|---|---|---|---|---|
| Topology | 10 | 1 | 1 | 2 | 2 | 4 |
| Network Size | + | + | + | + | + | + |
| Physical Links | - | - | - | + | - | + |
| Buffer modeling | + | + | + | + | + | + |
| Routing algorithm | 20 | 8 | 6 | 3 | 3 | 6 |
| Area analysis | - | - | - | - | - | - |
| Power analysis | - | - | + | - | + | - |
| Throughput analysis | + | + | + | + | + | + |
| Latency analysis | + | + | + | + | + | + |
| Heat map analysis | - | - | - | - | + | + |
| Synthetic traffic | + | + | + | + | + | + |
| Custom traffic | - | - | + | + | - | + |
| Packet injection ratio | + | + | + | + | + | + |
| Flit injection ratio | - | - | - | - | - | + |
| Mixed language | - | - | - | - | + | + |
| Cycle accurate | - | + | + | + | + | + |

+ : Supported, - : Not supported

plication does not have complete data available before packet is assembled. It supports rich set of utilities to evaluate NoC which includes latency, throughput, heat map and others. All the simulators studied in this thesis has capability to apply synthetic traffic over the NoC. Few simulators including NoCExplorer has advantage of applying custom traffic over the NoC. Support for more topologies and routing algorithms should be considered for future work. Area estimation and analysis is not possible in simulators we have studied in this thesis. With support for mixed language simulation it is possible to do early estimation of area and power as future work. Since NoCExplorer is cycle accurate simulator, there is a penalty of increased simulation run time.

# NoCExplorer simulation and results

# 4

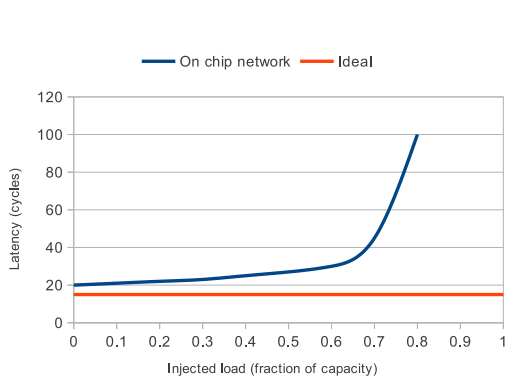
In this chapter we show and analyze the results of simulations that have been performed using NoCExplorer. First we explain qualification of NoCExplorer by correlating with published journals and papers for selected Network on Chip (NoC) architectures. Researchers have proposed various techniques and published results to compare NoC architectures. In our thesis we focus on evaluating NoCExplorer by comparing topology, routing algorithms and buffer sizing in routers and network interface. Then we show the impact of flit interval rate on the NoC performance in terms of latency and throughput. In the final section of the chapter we analyze execution time of NoCExplorer by profiling the code.

## 4.1 NoCExplorer evaluation

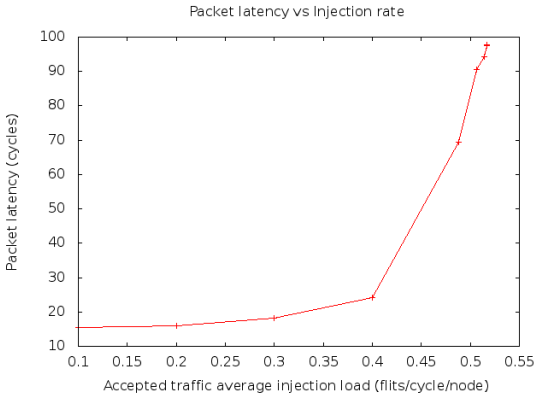
In Section 2.3 we explained in detail about the work done by Kumar *et al.* [15] and latency analysis results published as shown in Figure 4.1a based on 65nm CMOS technology library. To evaluate the correctness of NoCExplorer we mimic the setup of NoC architecture in our experiment. In the publication all the details required to configure a NoC in NoCExplorer is not available. So, we made certain assumption in our experiments like data size, simulation time and others. Hence while comparing results, we focus on latency trend in the network. A 7x7 mesh topology, xy routing and eight virtual channels based NoC was created using NoCExplorer with application of random synthetic traffic. Injection load which represents amount of flits entering into the network per cycle for each node (flits/cycle/node). Multiple simulation runs and measurements were done for varying injection load.

By analyzing the results we found NoCExplorer results correlated and similar trends were observed with increasing injection load into the NoC as shown in Figure 4.1. Till the network saturates there is a steady increase in packet latency. Once the network saturates there is an exponential increase in packet latency due to congestion in the network. Since we made certain assumptions in our NoC configuration due to non-availability of information, we do not compare latency numbers with published work.

We created another set of experiments based on study done by Hu *et al.* [25]. We setup a mesh topology (5x5) with XY, west first and south last routing algorithms for uniform random traffic similar to NoC proposed by Hu *et al.* [25]. Simulation results of the NoC using NoCExplorer is shown in Figure 4.4. XY routing performs better than partially adaptive based routing algorithms for uniform random traffic. Similar results were published by Hu *et al.* [25], Glass *et al.* [26], Mello *et al.* [27] and Chiu *et al.* [28]. Partially adaptive routing algorithms can potentially speed up the time to deliver individual packets but globally its performance is poorer than XY routing algorithm. In




(a) Latency analysis by Kumar *et al.* [15]




(b) Latency analysis using NoCExplorer

Figure 4.1: Comparison of latency analysis in NoC

XY routing algorithm based NoC, routers at center of the network takes $\sim$ 20 cycles per packet. Partially adaptive based routers at center of the NoC took $\sim$ 42 cycles per packet. Hence we know partially adaptive routing algorithms tends to concentrate traffic in the center of the network, increasing the number of blocked paths. Glass *et al.* [26] suggest that by reducing the number of turns in the routing algorithm may reduce blocking and improve performance. We also simulated the NoC for various network sizes and buffer sizes. All the results reflects similar characteristics as shown in Figure 4.2. Hence these experiment improved the confidence on correctness of the results using NoCExplorer.

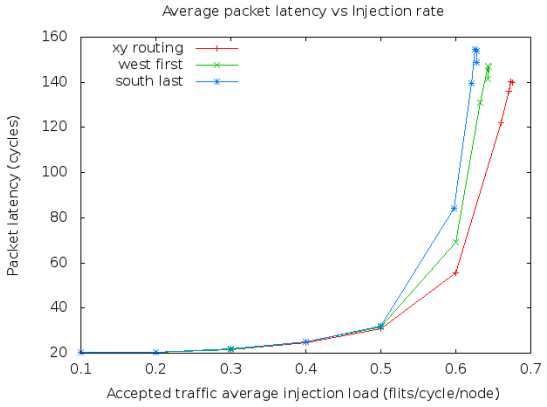
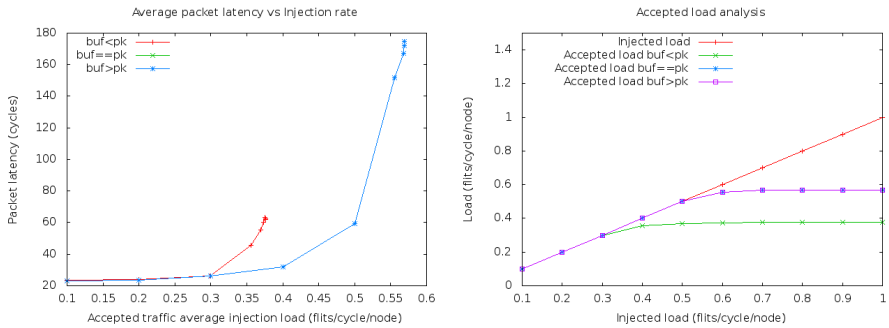



Figure 4.2: Average packet latency analysis for multiple routing algorithms

In previous experiment we were able to evaluate three routing algorithm and compare its characteristics using NoCExplorer. We updated the NoC architecture to have torus topology with torus XY routing algorithm. As mentioned in Section 2.2.1 we know that torus XY routing algorithm can cause deadlock in the NoC. In our NoCExplorer we have not implemented deadlock recovery techniques in the NoC, which could be considered for future updates. When we analyze the results generated by NoCExplorer, the results

for torus xy routing is not correct after the network saturates. This is because in NoCExplorer we only evaluate results for received packets in slave network interface. We do not evaluate missing packets in the NoC which would have stalled due to deadlock because of routing algorithms. Hence, we could not evaluate complete NoC characteristics for routing algorithms which might cause deadlock in the NoC after network saturates. Individual packets which has completed data transmission can still be analyzed which is not impacted by deadlock issues.

Design of buffer depth in router design has impact on area and power consumption. We used same NoC architecture proposed in previous evaluation of NoCExplorer to analyze the impact of buffer depth in NoC. We applied one cycle flit interval with packet size of 24 bytes which would translated to 8 flits (including head and tail flit). Multiple simulation run were done with three buffer depth values (4, 8 and 12) using NoCExplorer. This is done to emulate three scenarios when packet is transmitted through a router which are *a*) Buffer depth > packet size *b*) Buffer depth == packet size *c*) Buffer depth < packet size. In Figure 4.3 we show the results generated and analyzed from NoCExplorer. We can observe that the latency and throughput remains same for three buffer depth values until the network saturates. And when buffer depth is equal or more than packet size the performance of the NoC is same. When buffer depth is less than packet size the network saturates faster and there is gap of 40% in throughput compared to other two scenarios as shown in Figure 4.3b. Hence for given set of NoC architecture we can apply a thumb rule to restrict packets size equal or less than buffer depth to achieve optimal performance in the NoC. Hu *et al.* [29] and Mello *et al.* [30] have proposed and shown similar characteristics on the impact of buffers and virtual channels in router architecture.




(a) Average packet latency analysis (b) Accepted load analysis

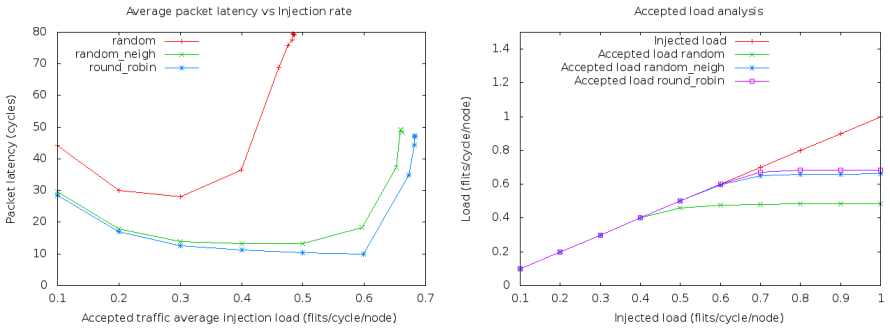
Figure 4.3: Latency and load analysis for multiple buffer depths

## 4.2 Flit interval selection

In Section 3.3.2 we explained modeling of flit interval rate when packet is transmitted from master network interface to the connected router. In related work discussion done in this thesis, none of them have highlight the impact of flit interval rate. Majority of the work assume complete data is available before packet is generated. Flit interval rate have

an impact on the NoC latency and throughput performance based on injection load into the network. We configured multiple NoC architectures using NoCExplorer to evaluate the impact of flit interval rate. In this section we will analyze one NoC architecture in detail since all the results obtained have similar correlation in latency and throughput. A 5x5 mesh topology with xy routing, single physical link, four virtual channels has been created as a base setup. Random, random neighbor and round robin neighbor synthetic traffic was applied on the NoC. Four flit interval rate one cycle, fixed flit, random flit, worst case flit interval supported in NoCExplorer was applied on the NoC.

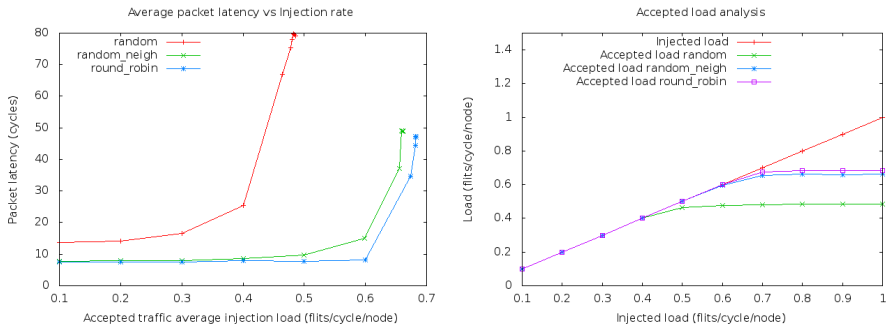
In Figure 4.4 and Figure 4.5 shows the NoC latency and load analysis for random flit interval and one cycle flit interval generated using NoCExplorer as explained in Section 3.3.2. From the Figure 4.4b we know random application of traffic saturates the network at 0.4 injection load. The other two traffic application saturates the network $\sim$ 0.6 injection load. With application of random flit interval we can observe packet latency at 0.1 accepted traffic injection load is highest till network saturates as shown in Figure 4.4a. By comparing it with application of one cycle flit intervals as shown in Figure 4.5a, packet latency reduces by more than 60% in all synthetic traffic scenario at lower injection load before network saturates. This shows even for localized synthetic traffic has an impact based on flit interval rate. The reduction in packet latency at




(a) Average packet latency analysis (b) Accepted load analysis

Figure 4.4: Latency and load analysis for random flit interval

lower injection load when one cycle flit interval is applied is due to less buffer back pressure and blocked routers in the NoC. Assuming packet injection rate is *tcycles* as shown in Figure 4.6, for one cycle flit interval mode will have routers and network interface resources allotted for 4 cycles. The routers and network interface resources will be available for other packets to be serviced for *tcycles-4* cycles. In random flit interval, routers and network interface resources will be available for other packets to be serviced for only $tr4$ cycles (shown in Figure 4.6). If two or more packets try to access the same resources as shown in Figure 4.6, one of the packet would be blocked till the tail flit of other packet passes through the shared resources. Hence, at lower injection load packet latency increases as flit interval increases in a packet.




(a) Average packet latency analysis (b) Accepted load analysis

Figure 4.5: Latency and load analysis for one cycle flit interval

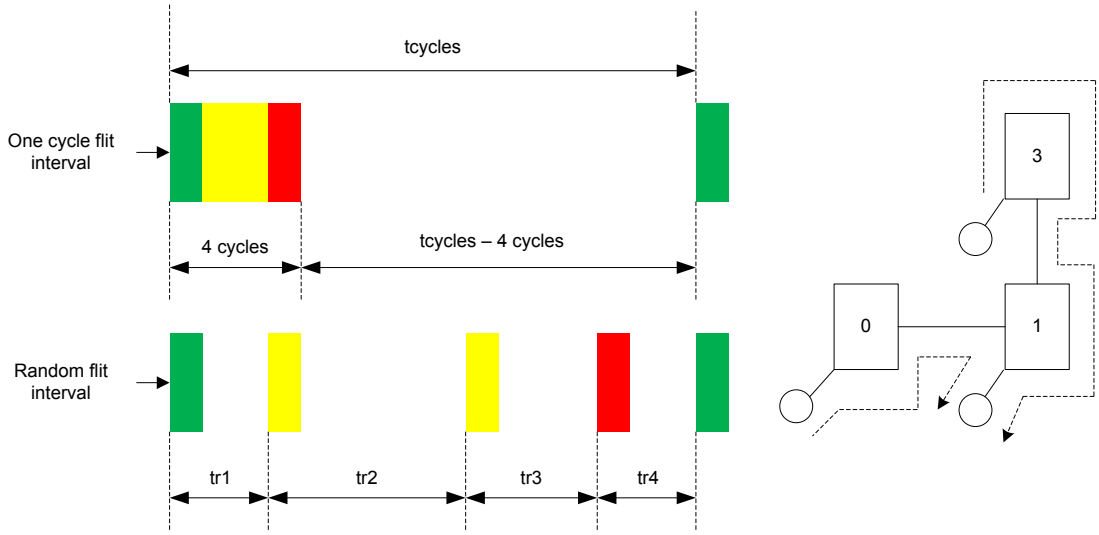



Figure 4.6: Flit blocking effect

## 4.3 Code profiling and run time improvements

Run time and utilization of compute resources is an important factor for usage of NoCExplorer. NoCExplorer should have low run time and maximum usage of compute resources provided to it. To evaluate the execution performance of NoCExplorer we reused the setup mentioned in Section 4.1. Gprof was used as profiling tool to profile the NoCExplorer. Figure 4.7 shows list of modules with run time $> 0.1\%$ of total run time. We can observe $\sim 50\%$ of run time is due to read and write of flit for every clock cycle. This is due to flit model as mentioned in Section 3.3.1 has to carry lot of information for working of the NoC and analysis purpose. Hence, it influence $\sim 50\%$ of run time in NoCExplorer. There is scope to optimize the modules and flit operations to improve the simulation run time. As an example, flit copy operation can be skipped if we optimize NoCExplorer to work only at packet level. But this would have impact on accuracy of the results.

| Time % | seconds | seconds | calls | name |
|---|---|---|---|---|
| 29.78 | 34.96 | 34.96 | 891855050 | flit::operator=(flit |
| 11.61 | 48.59 | 13.63 | 0 | pl_vc_alloc::process() |
| 10.96 | 61.46 | 12.87 | 0 | virtualchannel::rd_thread() |
| 7.92 | 70.76 | 9.3 | 315652724 | common::emptyFlit(flit&) |
| 6.53 | 78.43 | 7.67 | 0 | sc_core::sc_signal<flit |
| 4.87 | 84.15 | 5.72 | 0 | virtualchannel::status_thread() |
| 4.46 | 89.39 | 5.24 | 0 | route_compute::rc_thread() |
| 3.8 | 93.85 | 4.46 | 390661440 | sc_core::sc_signal<flit |
| 3.02 | 97.4 | 3.55 | 408581479 | sc_core::sc_signal<int |
| 2.4 | 100.22 | 2.82 | 0 | reg<flit>::write() |
| 1.37 | 101.83 | 1.61 | 0 | sc_core::sc_signal<sc_dt::sc_logic |
| 1.33 | 103.39 | 1.56 | 0 | slave_ni::receiveFlit() |
| 1.08 | 104.66 | 1.27 | 311041784 | Fifo<flit>::isEmpty() |
| 1.01 | 105.85 | 1.19 | 1325 | sc_core::sc_out<int>::~sc_out() |
| 1.01 | 107.03 | 1.18 | 426966500 | sc_core::sc_signal<flit |
| 0.98 | 108.18 | 1.15 | 0 | crossbar::process() |
| 0.97 | 109.32 | 1.14 | 258584483 | sc_core::sc_signal<flit |
| 0.93 | 110.41 | 1.09 | 0 | master_ni::injectFlit() |
| 0.78 | 111.32 | 0.91 | 0 | non-virtual |
| 0.72 | 112.17 | 0.85 | 0 | sc_core::sc_port<sc_core::sc_signal_inout_if<int> |
| 0.68 | 112.97 | 0.8 | 0 | sc_core::sc_port<sc_core::sc_signal_in_if<pkReq> |
| 0.55 | 113.61 | 0.65 | 0 | sc_core::sc_spawn_options::~sc_spawn_options() |
| 0.53 | 114.23 | 0.62 | 0 | non-virtual |
| 0.47 | 114.78 | 0.55 | 63168414 | sc_core::sc_signal<sc_dt::sc_logic |
| 0.41 | 115.26 | 0.48 | 0 | tf_manager::tf_stats() |
| 0.21 | 115.51 | 0.25 | 0 | non-virtual |
| 0.18 | 115.72 | 0.21 | 300499900 | Fifo<flit>::getFree() |
| 0.18 | 115.93 | 0.21 | 0 | virtualchannel::wr_thread() |
| 0.15 | 116.11 | 0.18 | 0 | sc_core::sc_signal<int |
| 0.14 | 116.28 | 0.17 | 0 | non-virtual |
| 0.14 | 116.44 | 0.16 | 0 | frame_dummy |
| 0.12 | 116.58 | 0.14 | 0 | non-virtual |
| 0.11 | 116.71 | 0.13 | 156384083 | sc_core::sc_signal<sc_dt::sc_logic |

Flit read and write operation at each clock cycle

Figure 4.7: Code profile

SystemC used for implementation in NoCExplorer has the adaptability for cycle and transaction level simulation and ability to model concurrent process. However SystemC libraries used in NoCExplorer supports only single thread simulation kernel which prevents it from utilizing multicore machines to speed up hardware simulations. Ezudheen *et al.* [31] present method to parallelize SystemC simulation kernel for fast hardware simulation on multicore machines. Kai *et al.* [32] show speed up in hardware simulation can be done by distribution of SystemC simulations without changing simulation kernel. We recommend these improvements should be integrated in NoCExplorer as future work.

# Application mapping on NoCExplorer

# 5

NoCExplorer has various capabilities to model a node to generate traffic for the Network on Chip (NoC). This can be effectively used to map applications, custom traffic scenarios to find performance characteristics of a NoC. In this chapter we explain Digital Audio Broadcasting (DAB) application transaction modeling and mapping onto many NoCs. Results from the experiments are shown and discussed in this chapter. We also show importance of application mapping over NoC is critical to achieve best performance results.

## 5.1 Digital Audio Broadcasting

DAB is a digital radio technology for broadcasting radio stations. DAB system was developed by eureka-147 project in UK in 1992. Compared to AM and FM analog broadcasts which has problems of interference and frequency selective fading the DAB system can offer CD quality sound without these issues [33]. The DAB broadcast signal is arranged into a transmission frame as shown in Figure 5.1. The DAB system provides a couple of transmission modes with varying parameters based on user requirements as shown in Table 5.1. For our experiments we would use *mode I* with transmission frame generated for every $96ms$ once. For NoCExplorer we used the DAB application transaction mod-

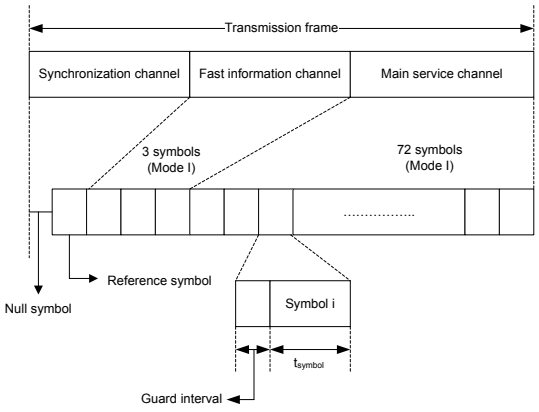



Figure 5.1: DAB transmission frame

eling based on a Synchronous Data Flow (SDF) graph [23] as an input. The input SDF as shown in Figure 5.2 represents transaction modeling of the application with defined tasks and communication between the tasks. The *src* task (which is connected to an ADC) outputs a frame at the rate of $96ms$ (as per Mode I shown in Table 5.1). The requirement of the application is that sum of computation and communication time for

Table 5.1: Transmission parameters of DAB system for different transmission modes [33]

| Mode | I | II | III | IV |
|---|---|---|---|---|
| Carrier number/ spacing | 1536/ 1kHz | 384/ 4kHz | 192/ 8kHz | 768/ 2kHz |
| Tframe | 96ms | 24ms | 24ms | 48ms |
| Tnull | 1.297ms | 324us | 168us | 648us |
| Tsymbol | 1.246ms | 312us | 156us | 623us |
| tsymbol | 1ms | 250us | 125us | 500us |
| tguard | 246us | 62us | 31us | 123us |

each task should not exceed $96ms$ so that *src* is not blocked. Blocking would cause buffer overflow and therefore distorted sound or a complete inability to decode. This would also ensure each frame of data is received from the ADC into the network.

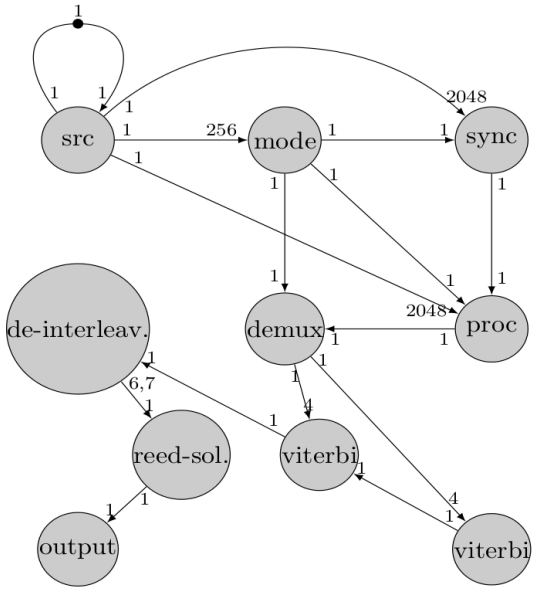



Figure 5.2: DAB SDF model

## 5.2 Application mapping on NoC

The mapping of application onto a NoC is one of the major challenges for design space exploration of NoCs. Application constraints provide bandwidth and latency requirements which needs to be satisfied in the NoC architecture. Many researchers have proposed various algorithms and techniques for efficient mapping of applications onto a NoC. Hu *et al.* [34] propose branch and bound algorithms to map nodes on mesh based NoC architectures to meet bandwidth constraints with minimum total energy consumption. They also propose a energy aware scheduling algorithm to map applications onto a NoC

in [35]. Murali *et al.* [36] show methods to split the traffic among various links in the network to meet bandwidth requirements of the task. They propose an algorithm for mapping with minimum path routing between communicating nodes in the network. The goal of the algorithm is to have least number of routers in the path between two communicating nodes. In our experiments we use this approach to map the application on different topologies using NoCExplorer.

As shown in Figure 5.2 the application has 10 tasks to be executed in the network. We make an assumption to map each task to a node in the network for our experiments. Only *src* and *output* tasks are combined to one node in our experiments. Designers has flexibility to merge multiple tasks into single node in NoCExplorer. Therefore with the assumptions we have, total number of nodes required for the application is nine nodes. These nodes should be efficiently mapped onto the network to achieve best Quality of Service (QoS).

In our experiments we created three different mappings based on minimum path routing between communicating nodes in the network [36]. They are

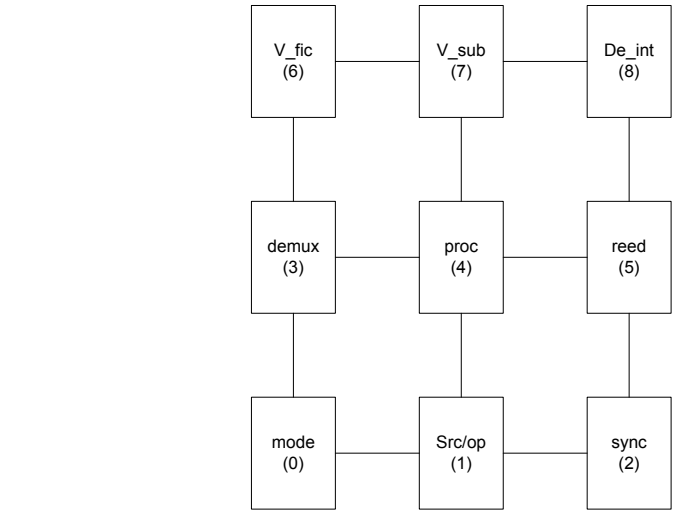
**Mesh:** Application was mapped on mesh topology as shown in Figure 5.3a. Tasks were assigned to nodes based on minimum path routing. All the communicating nodes have minimum hops possible as shown in Table 5.2.

**Torus:** In torus topology application was mapped as shown in Figure 5.3b. Similar to mesh topology mapping we have considered minimum hops possible for communicating nodes as shown in Table 5.2.

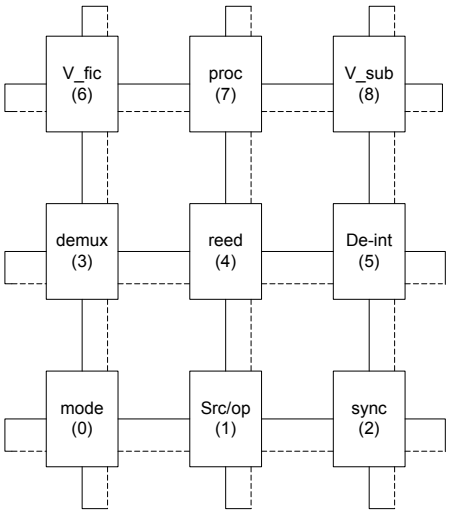
**Spidergon:** For spidergon topology we considered 'across first' and 'across last' routing algorithms to find minimum path routing for communicating nodes. Tasks were mapped on spidergon topology as shown in Figure 5.3c. Numbers mentioned as '()' in the Figure 5.3 are reference numbers used by NoCExplorer.

From mapping of applications onto the NoC we can derive the source and destination node when traffic is initiated. Using an SDF graph as input we need to calculate data size, bandwidth for sending the data and other parameters to model each node. In Table 5.2 we have shown hop count between communicating nodes. This is useful to determine minimum communication and computation time which in turn can be used to calculate lowest clock frequency possible for operation of the tasks assuming that there will be no congestion. For calculation purpose we assume following input parameters as mentioned in Listing 1. As explained in Section 3.4.2, we can calculate token/execution (amount of tokens being produced per execution of the task), required input tokens, cycle count required per task, data generated and received and bandwidth required for each task as shown in Table 5.3.

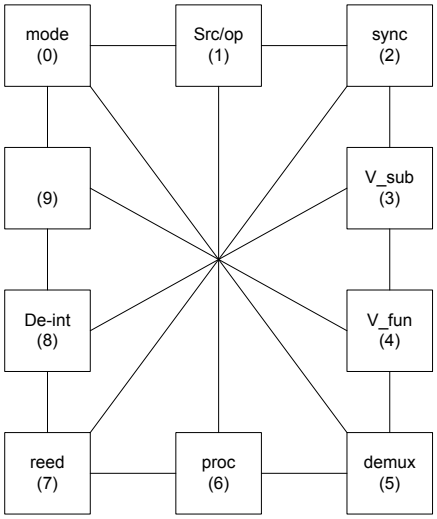
Bandwidth values was calculated, since it is required to model the node. With the information of required bandwidth and hop count from application mapping we can calculate communication time and computation time required for each communication tasks. Computation time is used to model wait time in the node before it can start its task of sending data into the network. When node is waiting to mimic computation time, it can still accept packets. This is to make sure computation and communication can be executed in parallel. Communication time represents required time to transmit the




(a) Mesh topology




(b) Torus topology




(c) Spidergon topology

Figure 5.3: DAB application modelling on topologies

**Listing 1** Input parameters for DAB modeling

```
1 frame = 29491.20 Bytes
Clock period of task = 10 ns
1 CIF = 7372.80 Bytes
1 codeword = 1134.28 Bytes
Burst size = 24 Bytes
NoC clock = 1000 ns
Packet size = 8 flits
```

Table 5.2: Minimum hop count for different mapping

| Source | Destination | Mesh | Torus | Spidergon |
|---|---|---|---|---|
| src/op | mode | 3 | 3 | 3 |
| src/op | sync | 3 | 3 | 3 |
| src/op | proc | 3 | 3 | 3 |
| mode | sync | 4 | 3 | 4 |
| mode | proc | 4 | 4 | 4 |
| mode | demux | 3 | 3 | 3 |
| sync | proc | 4 | 4 | 4 |
| proc | demux | 3 | 4 | 3 |
| demux | v_fic | 3 | 3 | 3 |
| demux | v_sub | 4 | 4 | 4 |
| v_fic | v_sub | 3 | 3 | 3 |
| v_sub | De-int | 3 | 3 | 3 |
| De-int | reed | 3 | 3 | 3 |
| reed | output | 4 | 3 | 4 |

Table 5.3: Bandwidth calculation for DAB application

| **Task** | **Tokens/ execution** | **Required input token** | **Cycle count (cycles)** | **Sent Data (Bytes)** | **Required data (Bytes)** | **Bandwidth (KBytes/ sec)** |
|---|---|---|---|---|---|---|
| src/op | 3 frame | 0 frame | 96ms | 88473.6 | 0 | 921.6 |
| mode | 3 frame | 1 frame | 2008715 | 88473.6 | 29491.2 | 4404.48 |
| sync | 1 frame | 2 frame | 15089 | 29491.2 | 58982.4 | 195448.33 |
| proc | 1 frame | 3 frame | 2058063 | 29491.2 | 88473.6 | 1432.95 |
| demux | 8 CIF | 2 frame | 1107799 | 58982.4 | 29491.2 | 5324.28 |
| v_fic | 1 CIF | 1 CIF | 34474 | 7372.8 | 7372.8 | 21386.55 |
| v_sub | 1 CIF | 2 CIF | 447304 | 7372.8 | 7372.8 | 1648.27 |
| De-in | 6.5 Codeword | 1 CIF | 40570 | 7372.8 | 7372.8 | 18173.03 |
| reed | 1 Codeword | 1 Codeword | 11269 | 1134.28 | 1134.28 | 10065.46 |

data for calculated bandwidth in an ideal point to point link. These calculations were done to find approximate value of NoC clock period, where application requirement of $96ms$ are satisfied. In our simulation runs NoC clock period was set based on calculated threshold value. In the Table 5.4 we shown latency and time required for operation of each task when it is applied on mesh topology. Also similar exercise was done for other topologies.

To measure the performance of the application over NoCs we would analyze the results in terms of frames. Sum of computation and communication time as shown in Figure 5.4 by each task should not be more than $96ms$ (input source rate). If the time taken is more than the input rate then we have deadline miss. For the analysis we consider deadline miss as an important parameter to assess the performance of the network.

Table 5.4: Latency calculation for DAB application mapped on mesh topology

| Src | Dst | Hop | Min latency (cycles) | Communication time (ms) | Computation time (ms) | Total time (ms) |
|---|---|---|---|---|---|---|
| src/op | mode | 3 | 12 | 44.24 | 0.00 | 44.24 |
| src/op | sync | 3 | 12 | 44.24 | 0.00 | 44.24 |
| src/op | proc | 3 | 12 | 44.24 | 0.00 | 44.24 |
| mode | sync | 4 | 14 | 51.62 | 20.09 | 71.71 |
| mode | proc | 4 | 14 | 51.62 | 20.09 | 71.71 |
| mode | demux | 3 | 12 | 44.24 | 20.09 | 64.33 |
| sync | proc | 4 | 14 | 17.21 | 0.15 | 17.36 |
| proc | demux | 3 | 12 | 14.75 | 20.58 | 35.33 |
| demux | v_fic | 3 | 12 | 29.50 | 11.08 | 40.57 |
| demux | v_sub | 4 | 14 | 34.41 | 11.08 | 45.49 |
| v_fic | v_sub | 3 | 12 | 14.78 | 1.38 | 16.16 |
| v_sub | De-int | 3 | 12 | 14.78 | 17.89 | 32.68 |
| De-int | reed | 3 | 12 | 14.78 | 1.62 | 16.41 |
| reed | output | 4 | 14 | 17.47 | 2.93 | 20.40 |

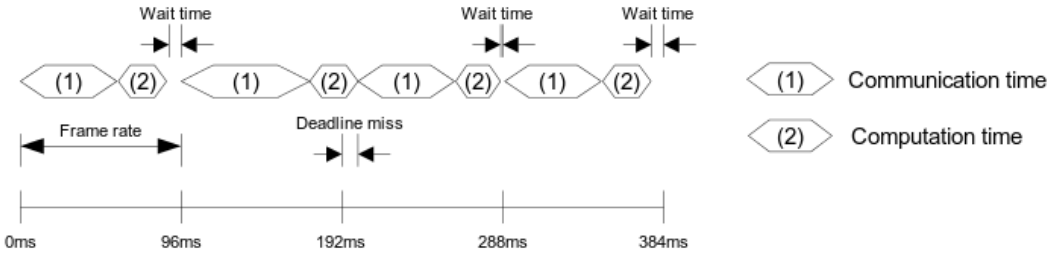



Figure 5.4: Timing diagram of deadline miss for DAB application

## 5.3 Results

In Section 5.2 we explained about different mappings we applied onto NoC for DAB application. Based on the four mappings, we made a list of NoC configuration with different routing algorithms to evaluate its performance as shown in Table 5.5. Rest of the configuration parameters such as buffer depth of 8 words, 4 virtual channels, round robin arbitration and others are not changed.

Table 5.5: NoC configurations to evaluate DAB application

| Topology | Routing algorithm |
|---|---|
| Spidergon | Across first<br>Across last |
| Mesh | XY<br>West first<br>South last |
| Torus | Torus XY |

We simulated each NoC mentioned in Table 5.5 by injecting 20 frames of data from *src/op* node at the rate of 96$ms$. Each frame was divided into multiple packets, with each packet containing 24 bytes. It was done to keep the packet size less than or equal to buffer depth for optimal performance (based on results shown in Section 4.1). Python utility tools were used to measure communication time and computation time for each communicating tasks. To meet the application requirement, we measured frame latency of *src/op* node to observe if it takes more than 96ms to inject data into the network. If the node took more than 96$ms$ to inject data into the network, extra time taken is referred as deadline miss.

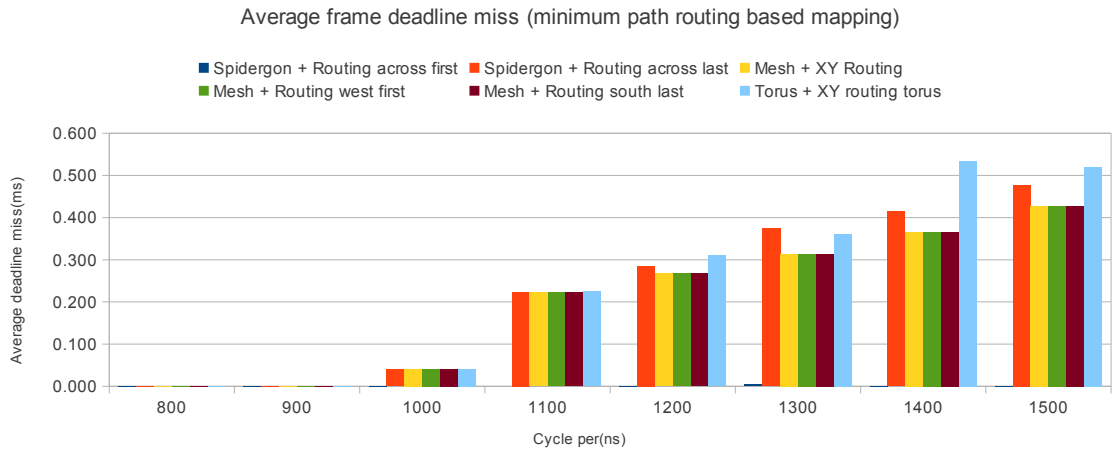



Figure 5.5: Average frame deadline miss

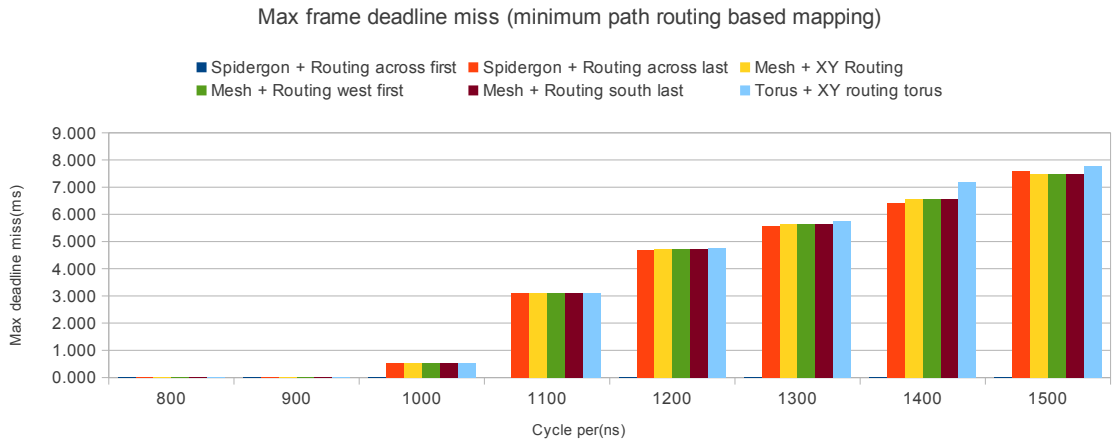



Figure 5.6: Max frame deadline miss

In Figure 5.5 we show average deadline miss for various NoC configurations. We can observe, upto clock cycle period of 900ns all the NoC configuration meet the application requirements. By further increasing the clock cycle period we found all the NoC configuration have approximately similar average deadline miss except spidergon topology with routing across first routing algorithm. It has least deadline miss value for given

constraints. The max deadline miss for each NoC configuration is shown in Figure 5.6. We could also observe that for given constraints mesh topology with various routing algorithms have similar results.

We analyzed the results in detail to find the reason for spidergon topology with across first routing to have best performance for given constraints. In Figure 5.7 we compared the average flit latency for all NoC configurations. We can observe spidergon topology with across first routing has the least average flit latency after clock cycle period of 900ns and above.

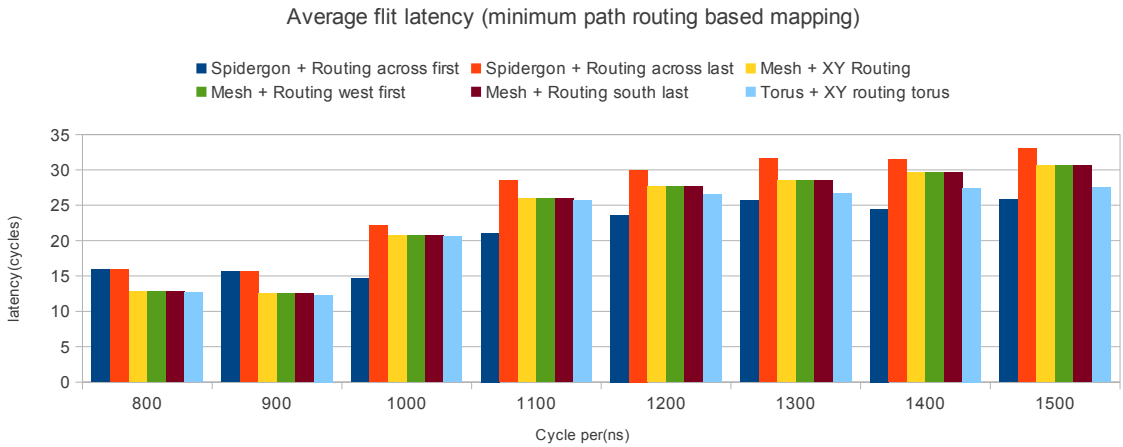



Figure 5.7: Average flit latency for minimum path routing based mapping

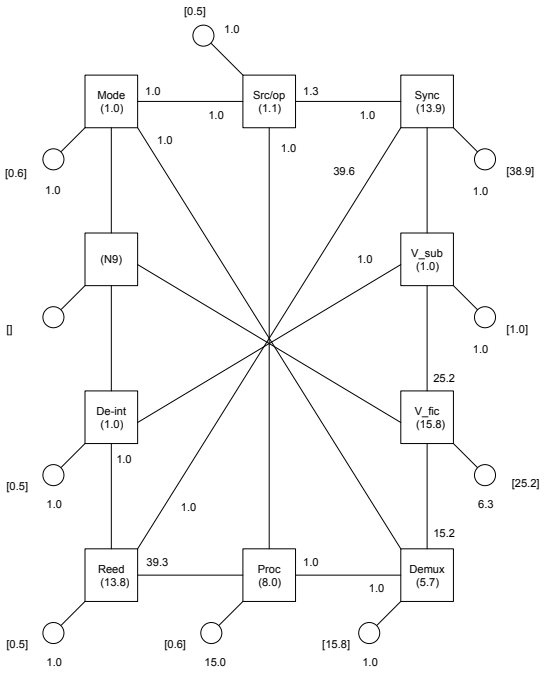



Figure 5.8: Router congestion for spidergon topology with across first routing

Router congestion visualization shows where bottlenecks are located and provides insights on hot spot location in the NoC. In Figure 5.8 we show router congestion in spidergon topology with across first routing algorithm. We can observe for *src/op* node overall router congestion is $\sim 1.1$ cycles/flit and link connected to router has congestion less than one cycle. By comparing with other NoC configuration as shown in Figure 5.9, the *src/op* node has router congestion more than 10 cycles/flit. Spidergon topology with across first routing algorithm has best performance due to least deadline miss, flit latency and low router congestion. Hence this NoC architecture is best suited for given application constraints.

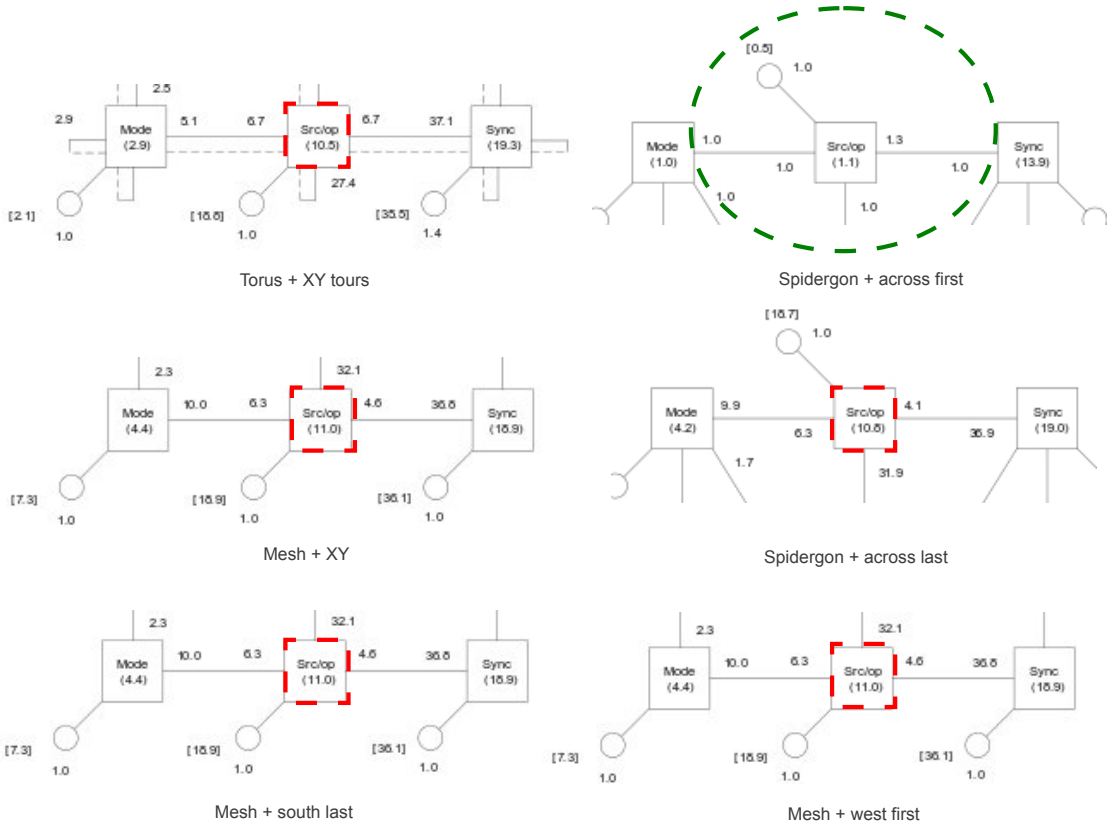



Figure 5.9: Router congestion comparsion for *src/op* node

Routing algorithms determines the route for packets to traverse in the network. Deterministic routing algorithms have set of fixed paths unlike adaptive routing algorithms which can by dynamic based on network congestion. Hence in deterministic algorithms like XY, turn based algorithms and others, 100% usage of links in the network might not be possible for mapped application. For the DAB application mapped on the multiple NoC configuration we found only 40% to 60% of links being utilized as shown in Figure 5.10. Torus topology with torus XY routing has least link used and mesh topology with routing south last has highest link used for given application. Hence in torus topology few links have utilization $> 40\%$ due to uneven traffic distribution as shown in Figure 5.11. To achieve 100% links being used in the network designers can change

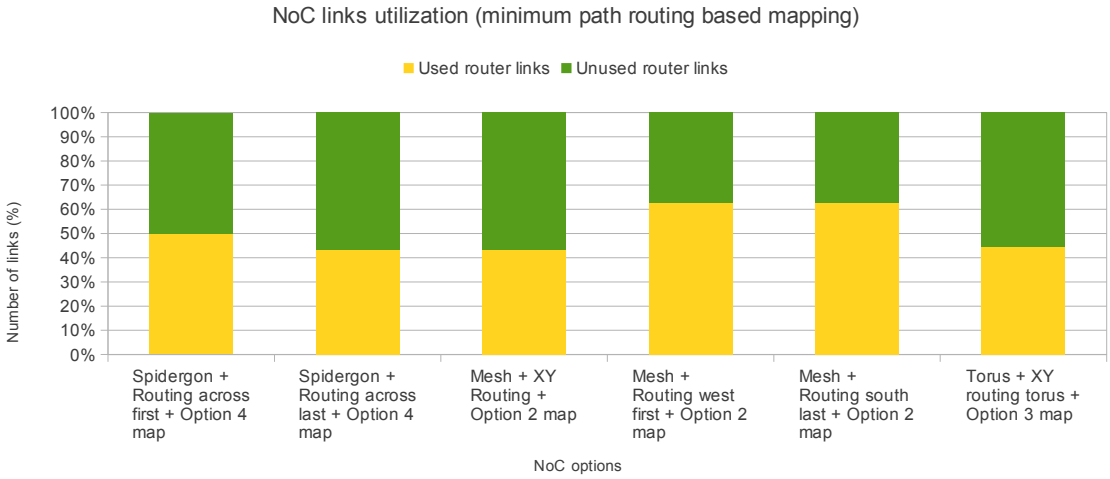
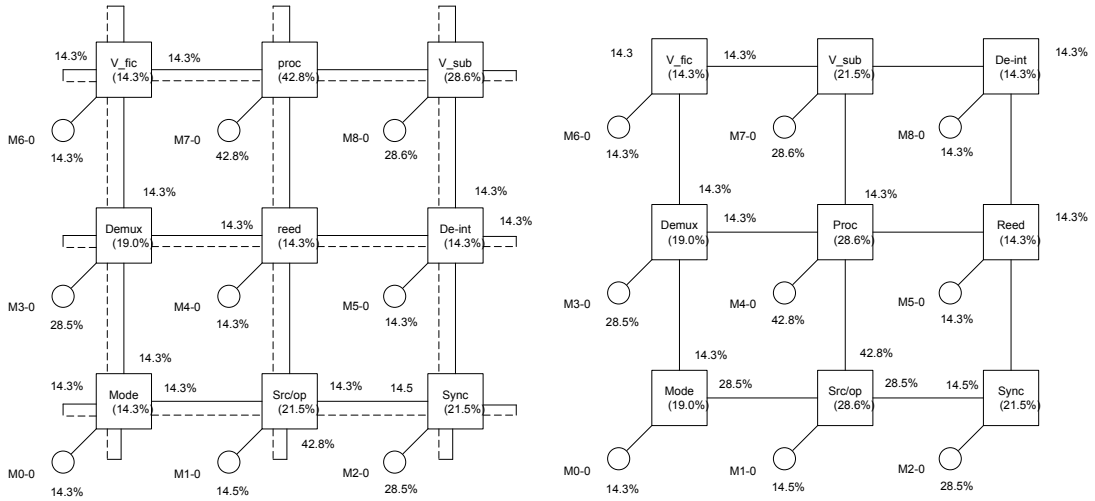



Figure 5.10: Link utilization in NoCs for DAB application

routing algorithm for even traffic distribution or remove unused links in the topology. By optimizing link usage and removing unused links, area can be saved in the silicon.




(a) Torus topology with torus XY routing (b) Mesh topology with routing south last

Figure 5.11: Link utilization in NoC

As mentioned in Section 5.2, mapping of application onto NoC is a one of the major challenges for design space exploration of NoCs. We explained usage of minimum path routing algorithm for mapping DAB application transaction model and its performance results. We observed spidergon topology with routing across first algorithm had best performance for given constraints. To highlight the importance of application mapping we used spidergon topology node configuration and changed topology connection to mesh and torus. Simulation and analysis of results as shown in Figure 5.12. Impact of routing algorithm on NoC performance is visible on mesh topology. NoCExplorer allows

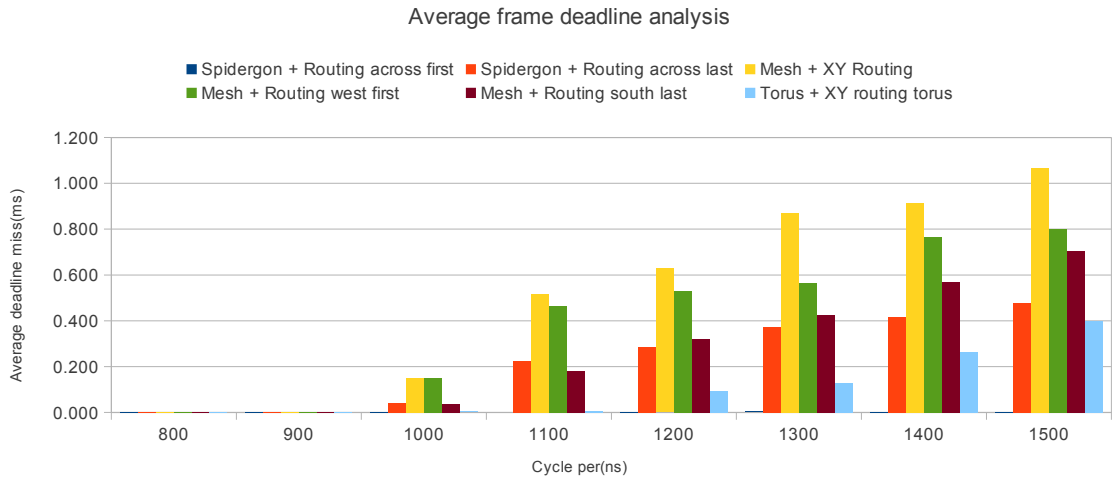



Figure 5.12: Average frame deadline miss for sub-optimal mapping

a designer to see the impact of sub-optimal application mapping over NoC architectures.

# Conclusion and recommendations

# 6

## 6.1 Conclusion

In this thesis we proposed a Network on Chip (NoC) characterization framework for design space exploration called NoCExplorer. The NoCExplorer is a cycle accurate simulator, developed using SystemC. It supports various techniques and architecture block combinations to generate a NoC. The framework also supports utility scripts based on python to analyze the results generated by NoCExplorer to assess the performance of the NoC.

Contribution of the thesis is as follows

- A framework to analyze the performance of a NoC.
- Evaluation of NoCExplorer with seven published journals and papers to build confidence on correctness of the simulator.
- A detailed comparison of the developed NoCExplorer with available open source simulators.
- An application specific design space exploration of different NoC architectures to identify communication bottlenecks of the application.
- A generic framework to enable automatic generation of parametrized NoCs with designer specific characteristics and the ability to easily include custom modules (SystemC or VHDL).

In the following we will discuss these contributions in more detail.

**NoCExplorer:** NoCExplorer is a set of tools which is part of a framework to perform design space exploration of NoC. It has capability to configure network size, physical links and others. Detailed buffer modeling can be done to set the buffer depth and number of buffers in a virtual channel. With parameters to control injection of packets into the network, NoCExplorer has advantage over other simulators to control even flit injection rate into the network. NoCExplorer supports individual blocks in the NoC to be synthesized or replaced with register transfer level VHDL code for mixed language simulations. Each node in the NoCExplorer supports detailed configuration to apply synthetic, Synchronous Data Flow (SDF) based traffic and custom traffic. It can transport real data at bit level via the network. It provides a rich set of performance parameters to analyze a NoC like packet or flit latency, throughput, injection load, link utilization and others. It has capability to generate heat maps for visual representation of link utilization and router congestion in the NoC.

**NoCExplorer evaluation:** A comparison with available open source simulators has been done to show how NoCExplorer provides larger selection of simulator parameters

and capabilities. NoCExplorer was correlated with seven journals and publications to prove the correctness of the results. A number of synthetic traffic patterns were mapped onto a set of NoC architectures and simulated. Results from the experiments showed cycle interval between flit injection into the network impacts the NoC performance. An increase in interval between flit injection within the packet significantly increases the latency of the NoC. The ratio of increase in latency varies for different NoCs under each set of synthetic traffic. Using NoCExplorer we also found routing algorithms which are prone to deadlocks which are difficult to evaluate. This is because measurements were incomplete due to stalled packets in the network due to deadlocks, and also eventually network stops accepting packets due to buffer backpressure. Performance of such routing algorithms can still be evaluated if we introduce deadlock recovery mechanisms into NoCExplorer as discussed in the recommendations.

**Application mapping on NoCExplorer:** The Digital Audio Broadcasting (DAB) application transaction model was used to show the capability of NoCExplorer to do early design space exploration of NoCs based on SDF graph. Selected set of NoC architectures were used to map the DAB application, which were simulated and results were analyzed using NoCExplorer. Stressing the NoC by reducing its clock frequency by a factor of 100x of core operating frequency, one of the spidergon based NoC with *across first* routing algorithm showed best results for the given application constraints.

**NoCExplorer synthesis:** A generic framework has been proposed to enable automatic generation of parametrized NoC with designer specific characteristics. SystemC based modular blocks in NoCExplorer supports easy addition and replacement to configure a NoC. It has been designed to be compatible with hardware modules so that it can be replaced with register transfer level VHDL.

## 6.2 Recommendations

NoCExplorer code profiling and detailed run time analysis was presented in Section 4.3. There is scope to optimize the modules and flit operations to improve the simulation run time. As an example, flit copy operation can be skipped if we optimize NoCExplorer to work only at packet level. But this would have impact on accuracy of the results. Also simulation run only on single thread simulation kernel which prevents it from utilizing multicore machines to speed up hardware simulations. For future work we should look at improving simulation run times by utilizing the potential of multicore machines by distribution of simulation work or by using multi threading support in simulation kernel and others.

Current work cannot evaluate routing algorithms which prone to deadlocks after network saturates. It is because in NoCExplorer there is no support for deadlock recovery mechanisms in routers and network interfaces. For future work if we include recovery mechanisms, routing algorithms options can be expanded and it can be used for evaluation even after network saturates.

Area and power estimation is possible by including the support for technology libraries and models in the framework. SystemC modular blocks can by synthesized using high language synthesis tools to get an estimate of gate count. Technology libraries have area and power information for individual gates. This information can be processed to

help designers for early estimation of area and power required to implement the NoC architecture.

Hardware architects or application developers can use this framework to integrate in platform simulation framework. It would combine computational blocks and interconnect architecture of a Multi Processor System on Chip (MPSoC) to do early design space exploration.

# Bibliography


[1] S. Pasricha and N. Dutt, *On-chip communication architectures: system on chip interconnect*. Morgan Kaufmann, 2010.

[2] M. Coppola, M. D. Grammatikakis, R. Locatelli, G. Maruccia, and L. Pieralisi, *Design of cost-efficient interconnect processing units: Spidergon STNoC*. CRC press, 2008.

[3] L.-S. Peh, S. Keckler, and S. Vangal, "On-chip networks for multicore systems," in *Multicore Processors and Systems*, ser. Integrated Circuits and Systems, S. W. Keckler, K. Olukotun, and H. P. Hofstee, Eds. Springer US, 2009, pp. 35–71. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-0263-4_2

[4] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004.

[5] V. Rantala, T. Lehtonen, and J. Plosila, *Network on chip routing algorithms*. Citeseer, 2006.

[6] A. Agarwal, C. Iskander, and R. Shankar, "Survey of network on chip (noc) architectures & contributions," *Journal of engineering, Computing and Architecture*, vol. 3, no. 1, pp. 21–27, 2009.

[7] G. De Micheli and L. Benini, *Networks on Chips: Technology and Tools*, ser. Systems on Silicon. Elsevier Science, 2006. [Online]. Available: http://books.google.nl/books?id=IHHTmSBcoGIC

[8] P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *Computers, IEEE Transactions on*, vol. 54, no. 8, pp. 1025–1040, Aug 2005.

[9] S. Suboh, M. Bakhouya, S. Lopez-Buedo, and T. El-Ghazawi, "Simulation-based approach for evaluating on-chip interconnect architectures," in *Programmable Logic, 2008 4th Southern Conference on*, March 2008, pp. 75–80.

[10] L. Bononi and N. Concer, "Simulation and analysis of network on chip architectures: Ring, spidergon and 2d mesh," in *Proceedings of the Conference on Design, Automation and Test in Europe: Designers' Forum*, ser. DATE '06. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2006, pp. 154–159. [Online]. Available: http://dl.acm.org/citation.cfm?id=1131355.1131388

[11] M. Moadeli, A. Shahrabi, W. Vanderbauwhede, and M. Ould-Khaoua, "An analytical performance model for the spidergon noc," in *Advanced Information Networking and Applications, 2007. AINA '07. 21st International Conference on*, May 2007, pp. 1014–1021.

[12] J. Balfour and W. J. Dally, "Design tradeoffs for tiled cmp on-chip networks," in *Proceedings of the 20th Annual International Conference on Supercomputing*, ser. ICS '06. New York, NY, USA: ACM, 2006, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/1183401.1183430

[13] P. T. Wolkotte, *Exploration within the network-on-chip paradigm.* University of Twente, 2009.

[14] B. Zitouni and R. Tourki, "Design and implementation of network interface compatible ocp for packet based noc," in *Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2010 5th International Conference on*, March 2010, pp. 1–8.

[15] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: Towards the ideal interconnection fabric," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 150–161. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250681

[16] S. P. Cristinel Ababei, Partha Pratim Pande, "Noc simulators," http://networkonchip.wordpress.com/2011/02/22/simulators/, 2011, [Online; accessed 1-April-2014].

[17] A. B. Achballah and S. B. Saoud, "A survey of network-on-chip tools." *International Journal of Advanced Computer Science & Applications*, vol. 4, no. 9, 2013.

[18] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. Dally, "Booksim interconnection network simulator," *Online, https://nocs. stanford. edu/cgibin/trac. cgi/wiki/Resources/BookSim.*

[19] F. Fazzino, M. Palesi, and D. Patti, "Noxim: Network-on-chip simulator," *URL: http://sourceforge. net/projects/noxim*, 2008.

[20] A. T. Tran and B. M. Baas, "Noctweak: A highly parameterizable simulator for early exploration of performance and energy of networks on-chip," Tech. rep., VLSI Computation Lab, ECE Department, University of California, Davis, Tech. Rep., 2012.

[21] S. K. Mandal, N. Gupta, A. Mandal, J. Malave, J. D. Lee, and R. Mahapatra, "Nocbench: a benchmarking platform for network on chip," in *Workshop on Unique Chips and Systems (UCAS)*, 2009.

[22] N. C. Aline Mello, Alexandre Amory and F. Moraes, "Atlas - a noc generation and evaluation framework," https://corfu.pucrs.br/redmine/projects/atlas, 2010, [Online; accessed 1-April-2014].

[23] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept 1987.

[24] P. R. Schaumont, *A practical introduction to hardware/software codesign.* Springer, 2012.

[25] J. Hu and R. Marculescu, "Dyad: Smart routing for networks-on-chip," in *Proceedings of the 41st Annual Design Automation Conference*, ser. DAC '04. New York, NY, USA: ACM, 2004, pp. 260–263. [Online]. Available: http://doi.acm.org/10.1145/996566.996638

[26] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *J. ACM*, vol. 41, no. 5, pp. 874–902, Sep. 1994. [Online]. Available: http://doi.acm.org/10.1145/185675.185682

[27] A. V. de Mello, L. C. Ost, F. G. Moraes, and N. L. V. Calazans, "Evaluation of routing algorithms on mesh based nocs," *PUCRS, Av. Ipiranga*, 2004.

[28] G.-M. Chiu, "The odd-even turn model for adaptive routing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 11, no. 7, pp. 729–738, Jul 2000.

[29] J. Hu and R. Marculescu, "Application-specific buffer space allocation for networks-on-chip router design," in *Proceedings of the 2004 IEEE/ACM International Conference on Computer-aided Design*, ser. ICCAD '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 354–361. [Online]. Available: http://dx.doi.org/10.1109/ICCAD.2004.1382601

[30] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual channels in networks on chip: Implementation and evaluation on hermes noc," in *Proceedings of the 18th Annual Symposium on Integrated Circuits and System Design*, ser. SBCCI '05. New York, NY, USA: ACM, 2005, pp. 178–183. [Online]. Available: http://doi.acm.org/10.1145/1081081.1081128

[31] E. P, P. Chandran, J. Chandra, B. P. Simon, and D. Ravi, "Parallelizing systemc kernel for fast hardware simulation on smp machines," in *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, ser. PADS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 80–87. [Online]. Available: http://dx.doi.org/10.1109/PADS.2009.25

[32] K. Huang, I. Bacivarov, F. Hugelshofer, and L. Thiele, "Scalably distributed systemc simulation for embedded applications," in *Industrial Embedded Systems, 2008. SIES 2008. International Symposium on*, June 2008, pp. 271–274.

[33] H. Su, B. Chi, and Z. Wang, "System design considerations of highly-integrated dab receiver rf front-end," *Consumer Electronics, IEEE Transactions on*, vol. 51, no. 4, pp. 1319–1325, Nov 2005.

[34] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based noc architectures under performance constraints," in *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '03. New York, NY, USA: ACM, 2003, pp. 233–239. [Online]. Available: http://doi.acm.org/10.1145/1119772.1119818

[35] R. Marculescu and J. Hu, "Communication and task scheduling of application-specific networks-on-chip," *Computers and Digital Techniques, IEEE Proceedings*, vol. 152, no. 5, pp. 643–651, Sept 2005.

[36] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto noc architectures," in *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, ser. DATE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 20 896–. [Online]. Available: http://dl.acm.org/citation.cfm?id=968879.969207

# List of definitions

.. ...

# Routing Algorithm

# A

## A.1 XY routing

**Algorithm 1** XY routing

**if** $dest\ ID[x\ dir] > router\ ID[x\ dir]$ **then**
  $DIRECTION\ EAST$
**else if** $dest\ ID[x\ dir] < router\ ID[x\ dir]$ **then**
  $DIRECTION\ WEST$
**else if** $dest\ ID[y\ dir] > router\ ID[y\ dir]$ **then**
  $DIRECTION\ NORTH$
**else if** $dest\ ID[y\ dir] < router\ ID[y\ dir]$ **then**
  $DIRECTION\ SOUTH$
**end if**

## A.2 Routing west first

**Algorithm 2** Routing west first

**if** $dst\ ID[x\ dir] <= router\ ID[x\ dir] || dstID[y\ dir] == router\ ID[y\ dir]$ **then**
  *Apply XY routing*
**else if** $dest\ ID[y\ dir] < router\ ID[y\ dir]$ **then**
  $DIRECTION\ SOUTH\ ||\ EAST$
**else**
  $DIRECTION\ NORTH\ ||\ EAST$
**end if**

## A.3 Routing south last

**Algorithm 3** Routing south last

**if** $dst\ ID[y\ dir] <= router\ ID[y\ dir] || dstID[x\ dir] == router\ ID[x\ dir]$ **then**
  *Apply XY routing*
**else if** $dest\ ID[x\ dir] < router\ ID[x\ dir]$ **then**
  $DIRECTION\ WEST\ ||\ NORTH$
**else**
  $DIRECTION\ EAST\ ||\ NORTH$
**end if**

## A.4 Torus XY routing

**Algorithm 4** Torus XY routing

$m_i[xdir] = (destID[xdir] - routerID[xdir])\%max(x)$
$m_i[ydir] = (destID[ydir] - routerID[ydir])\%max(y)$
**if** $m_i[xdir] <= max(x)/2$ **then**
  $delta[xdir] = m_i[xdir]$
**else**
  $delta[xdir] = m_i[xdir] - max(x)$
**end if**
**if** $m_i[ydir] <= max(y)/2$ **then**
  $delta[ydir] = m_i[ydir]$
**else**
  $delta[ydir] = m_i[ydir] - max(y)$
**end if**
**if** $delta[xdir] > 0$ **then**
  $DIRECTIONEAST$
**else if** $delta[xdir] < 0 and delta[xdir] != max(x)$ **then**
  $DIRECTIONWEST$
**else if** $delta[ydir] > 0$ **then**
  $DIRECTIONNORTH$
**else if** $delta[ydir] < 0 and delta[ydir] != max(y)$ **then**
  $DIRECTIONSOUTH$
**end if**

# Flit model

# B

**Listing 2** Modelling of flit

```
// Source ID and Destination ID:
// [0] = X direction, [1] = Y direction for mesh based topology
// [0] = node number for non-mesh based topology
int src_id[ROUTER_ID_DIMENSION];
int dst_id[ROUTER_ID_DIMENSION];
// The flit type (FLIT_TYPE_HEAD, FLIT_TYPE_BODY, FLIT_TYPE_TAIL)
flitType flit_type;
// The sequence number of the flit inside the packet
int sequence_no;
// ID of the packet in the network
int packet_id;
// timestamp at when packet is sent from master network interface
sc_time in_timestamp;
// timestamp at when packet is received at slave network interface
sc_time out_timestamp;
// timestamp at when head flit enters router or network interface
sc_time head_timestamp;
// Data to be sent
sc_lv<DATA_WIDTH> flitData;
// Virtual channel number
int vcNum;
// Next stage router direction
int opPortDirection[ROUTING_ALGO_DIR];
// Current number of hops from source to destination
int hopNum;
// Physical link in master network interface flit was generated
int mni_phylink;
// Physical link in slave network interface flit was received
int sni_phylink;
```

# Latency and throughput analysis file

C

An example file shows results generated by NoCExplorer SystemC modules. It shows virtual channel utilization, latency and throughput of the network.

**Listing 3** Sample result of latency and throughput analysis

```
// Clock period
Clock period,800 ns
// Injected load configured
Injection load,0.1
// Virtual channel utilization (flits/fifo)
VCNUM-0,114897
VCNUM-1,114893
VCNUM-2,114890
// Latency
Average packet latency(cycles),22.9747040225
Min packet latency(cycles),11
Max packet latency(cycles),261
Average flit latency(cycles),15.9522288012
Min flit latency(cycles),4
Max flit latency(cycles),254.5125
Average flit(3 hop) latency(cycles),15.2345474594
Min flit(3 hop) latency(cycles),4
Max flit(3 hop) latency(cycles),197.875
Average flit(4 hop) latency(cycles),17.241121152
Min flit(4 hop) latency(cycles),6
Max flit(4 hop) latency(cycles),254.5125
// Injection load
accepted traffic average injection load (flits/cycle),0.132815221806
accepted traffic min injection load (flits/cycle),0.0854352638906
accepted traffic max injection load (flits/cycle),0.254993691292
ejected traffic average injection load (flits/cycle),0.130923837403
ejected traffic min injection load (flits/cycle),0.0849955457735
ejected traffic max injection load (flits/cycle),0.247574490158
```