UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

Electronics and Computer Science

System-Level Design Automation and Optimisation of Network-on-Chips in Terms of Timing and Energy

by

Ji Qi

Thesis for the degree of Doctor of Philosophy

September 2015

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
Electronics and Computer Science

Doctor of Philosophy

SYSTEM-LEVEL DESIGN AUTOMATION AND OPTIMISATION OF NETWORK-ON-CHIPS IN TERMS OF TIMING AND ENERGY

by Ji Qi

As system complexity constantly increases, traditional bus-based architectures are less adaptable to the increasing design demands. Specifically in on-chip digital system designs, Network-on-Chip (NoC) architectures are promising platforms that have distributed multi-core co-operation and inter-communication. Since the design cost and time cycles of NoC systems are growing rapidly with higher integration, system-level Design Automation (DA) techniques are used to abstract models at early design stages for functional validation and performance prediction. Yet precise abstractions and efficient simulations are critical challenges for modern DA techniques to improve the design efficiency. This thesis makes several contributions to address these challenges.

We have firstly extended a backbone simulator, NIRGAM, to offer accurate system-level models and performance estimates. A case study of developing a one-to-one transmission system using asynchronous FIFOs as buffers in both the NIRGAM simulator and a synthesised gate-level design is given to validate the model accuracy by comparing their power and timing performance.

Then we have made a second contribution to improve DA techniques by proposing a novel method to efficiently emulate non-rectangular NoC topologies in NIRGAM and generating accurate energy and timing performance. Our proposed method uses time-regulated models to emulate virtual non-rectangular topologies based on a regular Mesh. The performance accuracy of virtual topologies is validated by comparing with corresponding real NoC topologies.

The third contribution of our research is a novel task-mapping scheme that generates optimal mappings to tile-based NoC networks with accurate performance prediction and increased execution speed. A novel Non-Linear Programming (NLP) based mapping problem has been formulated and solved by a modified Branch and Bound (BB) algorithm. The proposed method predicts the performance of optimised mappings and compares it with NIRGAM simulations for accuracy validation.

Contents

List of Figures

List of Tables

Declaration of Authorship

I, Ji Qi , declare that the thesis entitled *System-Level Design Automation and Optimisation of Network-on-Chips in Terms of Timing and Energy* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as: Section 1.3

Signed:...

Date:..

Acknowledgements

I would like to sincerely appreciate my supervisor, Prof. Mark Zwolinski, for his patient and helpful guidance, support and supervision throughout my Ph.D research.

I am also grateful for the research facilities and environmental support provided by the School of Electronics and Computer Science (ECS), University of Southampton, throughout my time at Southampton. In addition, I wish to thank the people of the Electrical and Electronic Engineering (EEE) Group, including Yang, Wei, Alex, Ime, Kier and Fang, who have made Bay 1 such a memorable place to work in the past four years.

Finally, I would like to express my particular thanks to my parents and my girlfriend Lin, for their continuous love, support and encouragement. This thesis is dedicated to them, without whom none of this would be possible.

To my parents, and beloved Lin...

Nomenclature

$b(e_{i,j})$	Minimum Bandwidth of Communication from Tasks t_i to t_j [bit/sec]
$bw(l_{i,j})$	Available Link Bandwidth Capacity of Nodes n_i to n_j [bit/sec]
$comm(e_{i,j})$	Communication Volume from t_i to t_j [bit]
E_{bit}	Bit Energy [J]
$E_{B_{bit}}$	Buffer Bit Energy [J]
$E_{bit}^{n_i,n_j}$	Bit Energy from Nodes n_i to n_j [J]
E_{Cbit}	Bit Energy between IP Core and Switch [J]
$E_{dst-Cbit}$	Bit Energy between Switch and IP Core of Destination Node [J]
$e_{i,j}$	Communication from Tasks t_i to t_j
$E_{L_{bit}}$	Link Bit Energy [J]
$e(l_{i,j})$	Communication Bit Energy from Nodes n_i to n_j [J]
$E_{S_{bit}}$	Switch Bit Energy [J]
$E_{src-Cbit}$	Bit Energy between IP Core and Switch of Source Node [J]
$E_{W_{bit}}$	Wire Bit Energy [J]
hop_{n_i,n_j}	Hop Counts between Nodes i and j
$l_{i,j}$	Link Routing Communication from Nodes n_i to n_j
n_{dst}	Destination Node of a Communication Routing Path
n_i	Selected Network Node i
n_{src}	Source Node of a Communication Routing Path
$p_{i,j}$	Potential Minimal Routing Path from Nodes n_i to n_j
$P_{i,j}$	Communication Volume from Nodes n_i to n_j [bit]
t_{dst}	Destination Task of a Communication Flow
t_i	Selected IP Task i
$t(l_{i,j})$	Bit Timing Cost from Nodes n_i to n_j [sec]
t_{src}	Source Task of a Communication Flow
$T_{bit}^{n_i,n_j}$	Bit Timing Consumption from Nodes n_i to n_j [sec]

Abbreviations

ACO	Ant Colony Optimisation
ASIC	Application-Specific Integrated Circuit
BB	Branch and Bound
BBN	Bayesian's Belief Network
BC	Best Case
BE	Best Effort
BFT	Butterfly Fat Tree
BLS	Binary Level Simulation
CHMAP	Chain Mapping
CMAP	Constructive Mapping
CMP	Chip Multi-Processors
CPU	Central Processing Unit
CSM	Core-Switching Mapping
DA	Design Automation
DC	Design Compiler
DCS	Distributed Computing System
DOR	Dimension Order Routing
DRSI	Double-Router to Single-IP
DSM	Dynamic Spiral Mapping
DSP	Digital Signal Processing
DyAD	Dynamic Adaptive Deterministic
ECS	Electronic and Computer Science
EDA	Electronic Design Automation
EEE	Electrical and Electronic Engineering
EM	Euclidean Minimum
EPAM	Energy- and Performance-Aware Mapping
ESL	Electronic System Level
FC	Fixed Centre
FDSM	Full Dynamic Spiral Mapping
FF	First Free
FIFO	First-In First-Out
flits	flow control digits

GA	Genetic Algorithm
GS	Guaranteed Service
HDL	Hardware Description Language
HOL	Head-Of-Line
IC	Integrated Circuit
IDCT	Inverse Discreet Cosine Transform
ILP	Integer Linear Programming
inout	input/output
IQuant	Inverse Quantisation
ISS	Instruction Set Simulation
LBMAP	Link Based Mapping
LEC-DN	Lower Energy Consumption based on Dependencies-Neighbourhood
LP	Linear Programming
LTS	Legal Turn Set
MAC	Minimum Average Channel
MaXY	Minimally Adaptive XY
MB	Macroblock
mCSM	many-to-many Core-Switching Mapping
MILP	Mixed Integer Linear Programming
MMC	Minimum Maximum Channel
MPSoC	Multi-Processor System on Chip
MSB	Most Significant Bit
MV	Motion Vector
NF	Neighbour-aware Frontier
NI	Network Interface
NIRGAM	Network-on-chip Interconnect Routing and Application Modelling
NLP	Non-Linear Programming
NN	Nearest Neighbour
NoC	Network on Chip
NoWs	Networks of Workstations
OE	Odd-Even
PBB	Performance-aware Branch and Bound
PDSM	Partial Dynamic Spiral Mapping
PE	Processing Element
PL	Path Load
PMAP	Physical Mapping
PSO	Particle Swarm Optimisation
PT	PrimeTime
QoS	Quality of Service
RC	Random Frontier
RISC	Reduced Instruction Set Computing

RTL	Register-Transistor Level
SA	Simulation Annealing
SAF	Store-And-Forward
SA-TA	Simulated Annealing with Timing Adjustment
SBMAP	Sort Based Mapping
SLS	Source Level Simulation
SoC	System on Chip
SRMI	Single-Router to Multiple-IP
SRSI	Single-Router to Single-IP
SSM	Static Spiral Mapping
TBMAP	Traffic-Balance Mapping
UMA	Uniform Memory Access
VC	Virtual Channel
VCD	Value Change Dump
VLD	Variable Length Decoder
VOL	Video Object Layer
VOP	Video Object Plane
WC	Worst Case
WSN	Wireless Sensor Network
WSNsim	Wireless Sensor Network simulator

Chapter 1

Introduction

1.1 Introduction

1.1.1 On-chip Network Architectures

As the integration of digital electronic circuits continues to scale up, an increasing number of transistors have been encapsulated on a single chip to build up larger and more complex systems. This higher systematic complexity brings about new design challenges that used to be ignored or given minor concern in the design process. For example, the consideration of leakage power cost becomes more essential in designing such highly complex on-chip systems. These new design challenges require researchers and designers to investigate effective solutions.

Moreover, the growing integration of digital systems also enables fast development of new patterns for data processing and mutual communication. Since the functionality and amount of processing data in these highly integrated systems has increased dramatically, the traditional bus-based architecture is hardly adaptable to tackle the enormous data requests with sufficient utilisation of available resources. Consequently, new patterns and architectures have been investigated to improve the efficiency and effectiveness of data processing in such systems. Multi-core processing and intercommunication is one promising pattern that processes data tasks in a parallel manner and implements functional systems as an interconnection network. This distributed pattern can better utilise available resources to process data more efficiently than traditional bus-based architectures, which leads to revolutionary improvements in system performance. It also prompts the advent of novel microprocessor structures that are applied in many computer fields, including on-chip systems, which can be named as *Distributed Computing Systems* (DCS) in general.

DCS architectures simultaneously distribute tasks onto many cores or functional modules, and collaboratively process them with frequent mutual communications. In particular, such characteristics enable DCS to fulfil the huge amounts of data requests and high complexity of functional implementation in modern on-chip systems. However, researchers and developers have faced many new problems to take full advantage of DCS architectures and make full use of available resources for optimised system performance. A major challenge is the implementation of intercommunications in DCS architectures. Specifically, complex computations can be split and co-processed on many DCS subsystem units in a distributed or parallel mode, which alleviates the processing workload of each single unit. But the computational results at each unit have to be communicated with others for further processing and task accomplishment, which raises the data communication overheads in DCS. Compared to traditional bus-based architectures, it suggests a shift of modern on-chip architectural designs from a computation-centric to a communication-centric basis [6]. Hence, the intercommunications among DCS subsystems has become very influential on overall system performance, which requires huge efforts to increase the efficiency.

Multi-Processor Systems on Chip (MPSoC) [17] and *Networks on Chip* (NoC) [18] are typical on-chip DCS architectures that are widely accepted as the potential solution to the problem of costly interconnection communications. In particular, NoC is considered as a central platform to suitably accommodate the communication requirements of future complex System-on-Chip (SoC) designs. This on-chip network paradigm employs routers interconnected by communication channels between pre-designed building blocks such as programmable RISC (Reduced Instruction Set Computing) cores, DSP (Digital Signal Processor) and memory blocks. The network holds a large number of functional components that tackle the high system complexity to perform versatile network functions while consuming tolerable production cost and design time cycles. Typical NoC architectures have received high research interest from researchers since the beginning of the last decade, to explore their potential for implementing complex on-chip systems with efficient network communications, high system performance and tolerable production costs.

1.1.2 Design Methodology of NoC Architectures

With the proposal of novel NoC architectures for highly integrated on-chip systems, their implementation methodology also needs to update for the fulfilment of efficient design flow. Due to the increased circuit integration, the *RTL (Register-Transistor-Level) synthesis* design flow no longer satisfied the demands of design productivity growth for complex on-chip systems. It is also proving increasingly difficult to bridge this productivity gap by simply increasing system abstractions for the design automation community [19]. This is because of the integration of transistors as well as the number of factors that must be considered are increased for NoC systems. Hence, researchers started to subsume *behavioural synthesis* into existing RTL synthesis and to tightly couple the fresh synthesis flow with physical design to improve the design productivity. As per [20], such a **classic** design flow is as follows:

1. Investigate behavioural models for each link of the application design process.
2. Simulate applications on a test bench for testing the functional performance and environmental impact on developed models.
3. Validate the model performance and functionality on prototypes.
4. Deploy the application in a realistic environment and examine its practical performance.

This design method was believed to be a plausible solution for efficient development of NoC architectures and applications at the beginning of the last decade. However, as complex applications that required large-scale network resources with busy communications were implemented on NoC systems, potential demerits of this design flow for testing performance emerged. The tremendous costs of production overheads and overlong design cycles of iterative refinement for optimised NoC applications seemed formidable in this classic design flow, which led to low design productivity and tight time-to-market budget. Specifically, the cost for prototyping certain networks with complex applications by using the design flow was increasingly expensive but essentially needed by modern on-chip products. Additionally, the difficulty in making full use of available network resources for processing complex NoC applications is also growing and thus results in unaffordable design time cycles.

To overcome such demerits and further promote the design efficiency, alternative design methodologies started to investigate replacement of the classic design flow. New design methods are in demand not only to refine the design flow but also to balance the design cost and productivity for optimised system acquisitions. A new method that abstracts high-level behavioural models and establishes network platforms to emulate system performance at early design stages is a constructive attempt. This method introduced a higher level of functional modelling into the classic RTL design flow, which decides the suitable NoC components from a system perspective to avoid overmuch iterative refinement in subsequent levels of design flow and thus save design cycles. Apparently, the accuracy of captured high-level models will fundamentally determine the method's usefulness. As few existing Design Automation (DA) techniques were able to precisely simulate the entire performance of practical circumstances at this level during the last decade, the generated simulation results often showed large deviations to cause fatal errors in realistic scenarios, which led to this high-level method not being applicable enough for further implementation and manufacturing. But this system-level model abstraction was such a promising idea that it was believed to be the alternative design methodology for efficient NoC designs if more advanced DA techniques can be developed.

Researchers in [19] had detailed the major tasks and challenges of this **new** methodology for NoC implementation:

- In the initial step, an executable functional system specification needs to be derived to clarify what exactly the designed system needs to do. Simulation may be a significant tool to specify the main system functions.
- In the next step, a task graph of concurrent tasks has to be developed. Decisions on intercommunication requirements between those tasks are also necessary. This step is important to determine the specific resources needed for implementation.
- Once the task graph is produced, mapping the tasks onto NoC platforms is a must. In this step many boundary constraints, component resources and system problems are required to be tackled for an optimised architecture.
- If each task has been bound to a specific component resource, the system-level model abstraction is assembled and the model codes must be generated for functional validation and performance simulation.
- For an optimised NoC design, all these steps have to be considered together with functions and performance validated iteratively. The deployment of accurate estimation techniques is essential to help make design decisions and simulate model performance, which ensures an applicable design methodology.

As the technology of Computer-Aided Design (CAD) has advanced, it seems feasible to move the design process to higher abstraction levels [14] and precisely optimise system performance via software implementation. Accordingly, several research groups, such as [21] and [1], have gradually developed the refined design flow of NoC interconnects from system level to physical layout-level implementation in an automated way. Their work suggested that suitable high-level design and synthesis tools for NoC designs is a key element to benefit from NoC-based systems in nanometer-scale technologies [1]. Near the end of the last decade, such a design flow that applying advanced design automation techniques for NoCs started from system-level modelling to accelerate the design flow has been accepted by various researchers as a promising design methodology. Exploring the implications of the NoC paradigm at different levels for efficient synthesis designs of NoC interconnects are continually proposed [22], [23], [24]. In [1], a typical **state-of-the-art** design methodology of NoC systems from system-level modelling to physical-level implementation is given. The complete NoC design flow consists of *three* main phases:

1. In the first phase or Front-End Phase, key architectural features of NoC systems like network topology, traffic flow control and routing tables are determined. Design automation techniques are required in this phase to improve design efficiency.

2. In the second phase or Architectural Design Phase, the RTL codes of target NoC architectures are instantiated based on determined models, design specifications and parametric constraints.

3. In the third phase or Back-End Phase, the layout of generated NoC architectures is synthesised, placed and routed, and implemented. A number of simulations and emulations are given to validate system performance like clock, power and area.

1.1.3 Design Space Exploration of Current Methodology

The **state-of-the-art** design methodology of NoC systems introduced in [1] merely proved that it is currently feasible to completely automate NoC systems from high-level specification to physical-level implementation. The whole design methodology is far from perfect to efficiently customise desirable NoC systems. A wide design space has been left open at almost every step of the design flow for further exploration.

Specifically in the **first phase** of the given state-of-the-art design methodology, design objectives and resource constraints to desired NoC systems are initially specified, followed by the task allocation of NoC applications. The system-level modelling of candidate NoC architectures in which key network characteristics and parameters satisfy those conditions is then generated. In the topology generation, iterative analysis to all candidate networks with concerned metrics are performed based on pre-defined settings. Various traffic flows are routed on those network architectures for performance estimates in order to choose one particular architecture that optimises design objectives while satisfying all the constraints. The design steps of the front-end phase are given in Figure 1.1.

Since the accuracy and efficiency of *system-level model abstraction* are the most fundamental cornerstones in producing efficient and reliable NoC designs for contemporary complex applications, in this thesis we will explore potential design space at system level to carefully balance the modelling for improving the productivity of design flow. To be more specific, each step of *Design Phase* 1 in Figure 1.1 explores its design space, investigating better design methods by utilising advanced modern design automation techniques. If the efficiency of all steps in design phase 1 is increased, a fully optimised system-level NoC model can be generated quickly with accurate performance and characteristic estimates, which will accelerate subsequent design phases to improve design cycles of the whole design flow. Moreover, as the design complexity of NoC systems grows to tackle the increasing computation requirements of various applications, finding optimised high-level models from a number of NoC candidates is more difficult and time-consuming. Design improvement in Phase 1 steps also enables a much faster searching process that saves more design cycles for the entire system implementation.

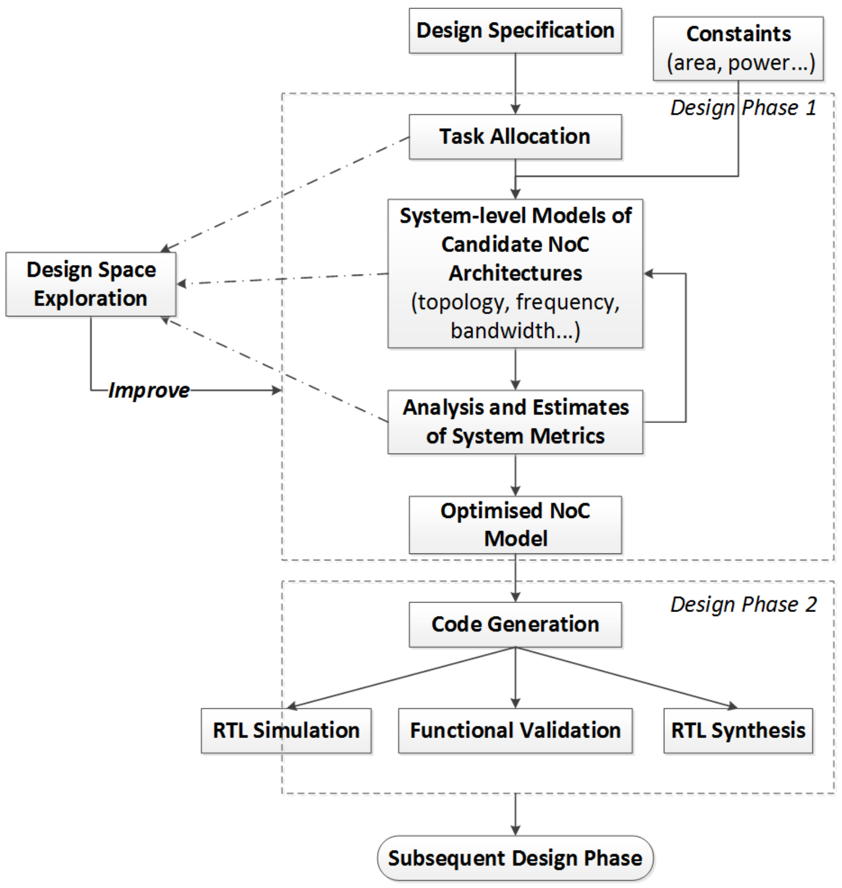


Figure 1.1: Explore Design Space in Phase 1 of State-of-the-art Design Flow [1]

Design space exploration in high-level modelling steps of the state-of-the-art design methodology raises several questions that further investigation will be conducted to solve them in this thesis:

- In the analysis step, are the performance estimates of system-level NoC models accurate enough to produce reliable optimised candidates for further design and implementation?
- In the high-level modelling step, are the NoC architectural and parametric models efficiently abstracted and flexible to represent desirable design specifications?
- In the task allocation step, can the current application tasks have better mapping schemes to balance the utilisation of available resources and optimised performance, such that both design efficiency and productivity are improved?

1.2 Research Objectives

Motivated by the fact that system level of the NoC design methodology has left wide design space open, this doctoral project aims to investigate advanced Electronic Design

Automation (EDA) techniques for efficiently developing accurate system-level models of NoC systems and optimised performance trade-offs in terms of energy and timing. The design space of each step in the initial phase of the state-of-the-art NoC design methodology is explored potential improvements to increase efficiency of the whole design flow. A better balance between model accuracy and design productivity is also considered. An extended simulator NIRGAM [25] is used in some design steps to help generate optimised models of NoC systems.

The expected improvements to current design methodology include to provide feasible task allocation schemes for versatile application requirements, accurate system-level behavioural models for efficient NoC design flow and reliable performance estimates in terms of energy and timing issues for optimised network candidates at early design stages. The developed techniques and methods at different design steps should be jointly applied to improve current design methodology for complex NoC applications. Figure 1.2 demonstrates the exact design space expected to explore at each step in the thesis.

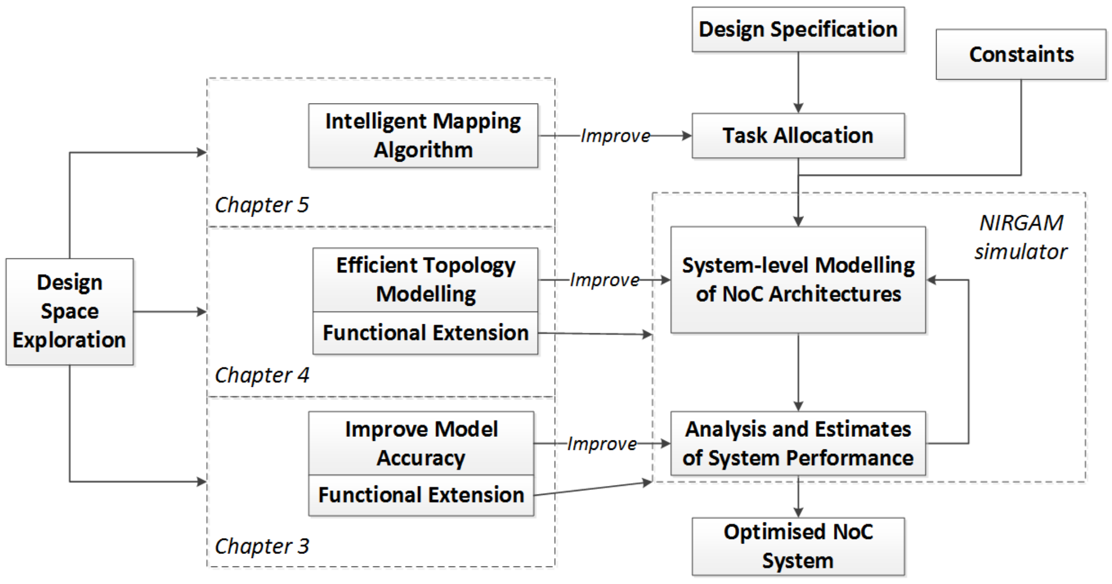


Figure 1.2: Research Objectives

As shown from the figure, the detailed research objectives of the research project are categorised as follows:

1. **Accurate Model Abstraction of System-Level NoC Architectures:** Before utilising EDA techniques for advanced NoC system designs and further performance evaluation, the accuracy of high-level behavioural models needs to be validated in Chapter 3 of the thesis. Functional and performance comparisons between high-level modelling and RTL synthesised designs using a traditional design method are helpful to improve the model accuracy. Since high-level models are developed in the NIRGAM simulator for accurate performance estimates, its

performance models need to be carefully calibrated with RTL synthesis technology library. Functional extension to the simulator is also expected if necessary.

2. **Efficient Modelling of Various NoC Topologies:**
   After the accuracy of system-level models is improved, offering more candidate NoC architectures with optimised performance while meeting design requirements are considered to explore potential design space. Most existing NoC design methodologies merely abstract basic NoC network architectures at the system level owing to the easy implementation of modular structures in such topologies. High-level abstractions of other NoC architectures used in specific applications may be achievable in the current design methodology but lack efficient modelling method due to the difficulty and expensive design cost of their architectural implementation. Hence, in Chapter 4 we will explore an efficient method to model non-rectangular and irregular NoC architectures at system level. Again, the NIRGAM simulator will be used as a platform for such modelling, which may require extra functionality.

3. **Optimised Task Mapping and Performance Trade-offs:**
   If both model accuracy and availability of system-level NoC architectures are improved, developing intelligent task mapping schemes for fast matching of optimised NoC systems is another space for improving the whole design methodology. Proper task mapping techniques with the optimisation of multiple design objectives are feasible to offer optimal trade-off performance for specific NoC applications. Due to the intrinsic nature of multi-objective problems, the optimisation techniques in high-level NoC designs are supposed to be application-specific, which leaves a large design space for performance improvement. Moreover, implementing specific task mappings for available NoC architectures may maximise the system performance, which offers optimal NoC matchings for various applications. Hence, we will explore potential design space of more intelligent and generalised mapping algorithms in Chapter 5.

1.3 Research Contributions

The contributions of this thesis are listed as follows:

- J. Qi, and M. Zwolinski, "Efficient Simulation and Modelling of Non-rectangular NoC Topologies," *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 1-4, 2014.

- J. Qi, M. Zwolinski1, V. Laxmi, and M.S. Gaur, "Area-efficient, Load-balancing, Non-rectangular Architectures for Networks on Chip," *Frontiers of Information Technology and Electronic Engineering*, (Under review).

- J. Qi, and M. Zwolinski, "NLP-based Application Mapping and Performance Prediction for Tile-based NoC Architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (Submitted).

1.4 Thesis Structure

Chapter 1 introduces background information for subsequent chapters. The concepts of on-chip networks and the development of its design methodology are included in the chapter. The motivation, research objectives and contributions of this project are given.

Chapter 2 discusses related research fields. Four areas of knowledge are mainly reviewed. The network on chip basics are initially proposed, followed by the opinions and concerns to current system-level designs of on-chip systems with advanced design automation techniques. Next, a survey of existing task mapping techniques for NoC systems is conducted. Finally a review of modern simulators for NoC designs is reported.

The next three chapters discuss the main contributions of this thesis. Chapter 3 presents research on the first objective, exploring the design space of improving high-level model accuracy for reliable system optimisation. A case study of developing asynchronous FIFOs as NoC router port buffers is undertaken to compare the performance accuracy of system-level abstract models with RTL synthesised designs. Models in the NIRGAM simulator are calibrated to achieve reliable simulation results. Other extensions and modifications to the simulator are also demonstrated.

Chapter 4 gives research towards the second objective, proposing an efficient method to emulate non-rectangular and irregular topologies at system level. This method models topological characteristics of different NoC architectures on a mesh network to accelerate the efficiency at this design step. A number of network topologies are supported by the proposed method, which enhances the functionality of the current NoC design methodology.

Chapter 5 introduces investigation of the third objective, exploring the design techniques for optimised task mapping allocation of specific applications to different networks under certain performance trade-offs. The proposed mapping algorithm provides better balance between the execution efficiency and mapping performance on various NoC architectures, improving the task allocation step with a more generalised and intelligent mapping algorithm that facilitates the yield of optimised NoC systems for further implementation.

The final Chapter 6 concludes the report and outlines plans for future work.

Chapter 2

Literature Review

Interconnection networks, like Multi-Processor System-on-Chip (MPSoC) or Network-on-Chip (NoC) systems, are increasingly popular in modern on-chip system designs for highly distributed communication demands. The efficient parallel computing and shared communication resources in such networks make them suitable to design complex on-chip systems with desirable performance. While enjoying these advanced features, a set of new problems such as data routing, flow control and topology [8] are also brought in and need to be tackled. Moreover, efficient and proper designs of such complex systems under tolerable time to market is harder to accomplish, which has also left large space for further exploration. In this case, comprehensive understandings of the NoCs and interconnection networks are an essential prerequisite for chip designers, which are initially introduced in this chapter from basic concepts to crucial characteristics. Next, investigation into existing opinions about system-level NoC modelling for performance estimation and optimisation at early design stages is conducted. Then, popular task mapping techniques for trade-off performance optimisation of NoC systems are surveyed to manifest current research progress at the task-allocation stage. Finally, a brief survey of modern network simulators for high-level modelling and performance estimation is presented in terms of their characteristics and extensibility, such that a backbone platform can be chosen for our further research.

2.1 Network on Chip Basics

2.1.1 Basic Concepts

As the chip feature size is continuously scaled down, more functional components are integrated onto a single die. Traditional bus-based architectures with a single processing unit are no longer scalable to task computing requirements since the bus structure becomes the performance bottleneck when more processors and functional units are integrated. The concept of network architectures is thus introduced and extensively adopted to loosen the bottleneck [2]. In modern chip design industries, multiprocessors dominate the parallel computing market and their architectures implement the interconnection networks with distributed control. According to [8] and [2], a network classification scheme is offered to categorise currently known interconnection networks into 4 major classes based mainly on the network topology, which are shared-medium networks, direct networks, indirect networks and hybrid networks (Figure 2.1). It is noteworthy that this classification focuses on existing networks and is not exhaustive. Other novel and innovative interconnection networks may constantly emerge as the technology further advances.

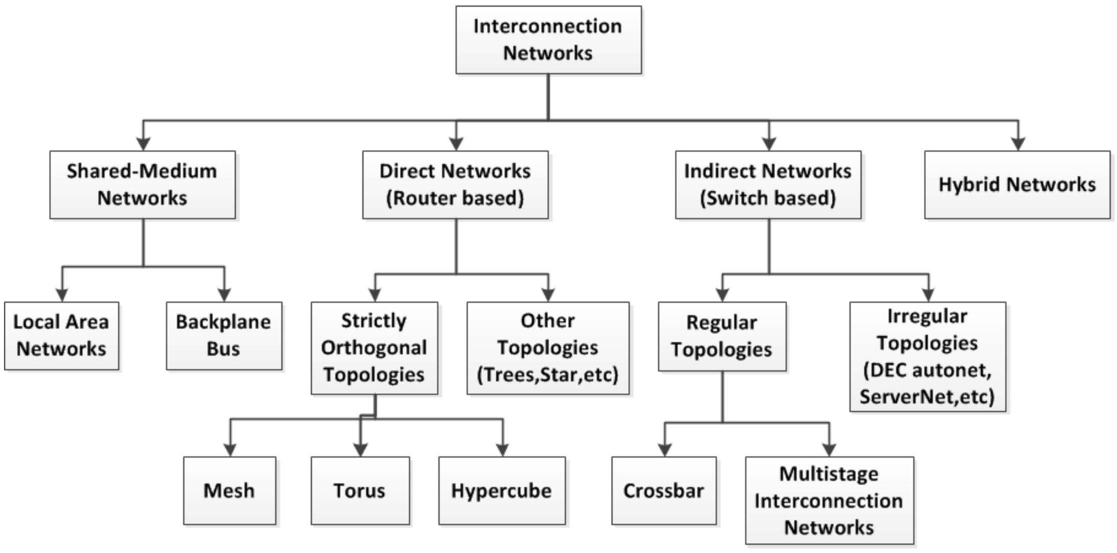

Interconnection Networks
Shared-Medium Networks
Direct Networks (Router based)
Indirect Networks (Switch based)
Hybrid Networks
Local Area Networks
Backplane Bus
Strictly Orthogonal Topologies
Other Topologies (Trees,Star,etc)
Regular Topologies
Irregular Topologies (DEC autonet, ServerNet,etc)
Mesh
Torus
Hypercube
Crossbar
Multistage Interconnection Networks

Figure 2.1: Four Classifications of Interconnection Networks [2]

As shown in Figure 2.1, the transmission medium in *shared-medium networks* is shared by all communicating components. An alternative to this approach, known as a *direct network*, has point-to-point links directly connecting each communicating unit to a subset of other communicating units in the network. In this case, any communication between non-neighbouring devices transmits data through several intermediate units. Instead, *indirect networks* connect those components by one or more switches that have point-to-point links to connect with each other. In this case, any communication between communicating units transmit data through switches. Finally, hybrid approaches are possible.

Since the primary investigation in this thesis focuses on improving NoC designs where low-power and efficient communications between wired-connection multiprocessors are mainly considered, concepts of direct, indirect and hybrid networks are reviewed later. The shared-medium network has a structural resemblance to a traditional bus structure and cannot mitigate the performance bottleneck of complex systems [8]. Most popular existing network topologies for MPSoC and NoC systems stem from direct and indirect network classes. Network topologies and their characteristics currently evaluated in this thesis are introduced subsequently.

2.1.1.1 Direct and Indirect Networks

In the beginning, several network concepts are introduced based on the graph presentation used in [8].

- **Node Degree** is the number of connecting channels from one node to its neighbours.
- **Network Diameter** is the maximum distance between any two nodes in the network.
- **Regularity:** means all the nodes in a regular network have the same degree.
- **Network Symmetry:** means a network is symmetric if it looks the same from each node.

As per the classification in [8], a **direct network** is a kind of networks that each network node directly connects to other nodes. A node in a direct network normally contain independent processors, memories/buffers and other functional components. These components can be designed programmable and reusable, making the direct network scales well when the network size increases [8]. A *router* is a common component in each node of direct networks. It is used to handle data communication among nodes [8]. For this reason, a direct network is also called *a router-based network*. Each router has a number of port channels to directly connect to neighbouring node routers. In an **indirect network**, data communication between each node pair is via *switches*. Thus, an indirect networks is also called a *switch-based network* [2]. Each switch also has a number of port channels that either remain open or connected for data transmission. These ports of different switches are interconnected with each other to communicate different processors [8].

A direct networks can be characterised by its topology, routing, and switching mechanism [11]. The network *topology* indicates how nodes are interconnected by router channels and are modelled by a graph [10]. An ideal direct network topology connects every node directly to each other. But the cost of such a topology will be too high if the network size gets large. Hardware limitations also lead to a constrained number of physical connections that a direct network node may have [8]. Hence, most of existing direct networks connect each of their nodes via hopping among intermediate nodes. A network *Routing algorithm* determines the network path by which a data packet reaches its destination from its source on a network topology [10]. A data packet is a data exchange unit for data transmission [8]. When a packet header reaches a node channel port, a *switching mechanism* is used to decide how and when the router should be set to forward the incoming data to its destination [8]. Buffer allocation and flow control techniques are important for an efficient switching mechanism. Enough available buffer space is needed to prevent drops of transmited data. A flow control technique establishes a conversation between the source and destination nodes to arrange a data flow [10]. Both techniques are significant to the network performance.

Indirect networks can be modelled by a graph $G(S, L)$, where S is the set of switches and L is the set of unidirectional or bi-directional links between the switches [3]. The switches that connect to processors can be the source or destination for data routings [26]. Data transmission between a node pair in an indirect network needs to cross the link between the source and its connected switch, and the link between the last switch on the routing path and the destination [8]. An indirect networks can also be characterised by its topology, routing, and switching mechanisms [11]. The topology determines how the node switches are interconnected by link channels and are modelled by a graph. Hardware limitations like the number of available pins and the available wiring area also constrain the number of physical connections an indirect network switch may have [10]. A suitable indirect network topology, routing or switching mechanism usually depends on specific hardware wiring and packaging constraints.

In modern chip design, researchers and developers usually consider direct and indirect networks as two different classes. However, recent developments in router architectures, algorithms, and switch designs have suggested a unified view of direct and indirect networks [8] [10]. In this view, direct networks can be considered as each switch in the network connecting to a node, while the switches of indirect networks can be considered as point-to-point links interconnected with one or more nodes. Hence, this unified view allows network design techniques, such as deadlock-free techniques, for one network class to be applicable to the other class. It is manifest that not all techniques can be directly used for both network classes. But such a unified view is a good attempt to properly describe some currently existing network topologies which have the characteristics of both network classes.

2.1.1.2 Development of Network on Chip

Before the emergence of Network on Chip concepts, Systems on Chip (SoC) are proposed as a typical kind of complex Integrated Circuit (ICs) which integrate major functional blocks of a digital electronic system onto a chip. Compared to traditional circuits, SoC implements a whole system on a chip to substantially improve the system performance and communication efficiency. However, its limitations are found as the IC technology continues to develop. One typical problem is the non-scalable global wire delay [6]. In former electronic circuits that the circuit scale is relatively small and the clock frequency is not as high as in modern SoC integrated circuits, wires carry signals through the whole chip and all timing restraints are met. But in modern complex SoC systems, the wire delay increases linearly or even exponentially with the scaling of transistor size, which may cause the wire delays to exceed an entire clock cycle. Although First-In First-Out (FIFO) structures like Bi-FIFO, Global bus 1 and 2 and Crossbar Switch bus architectures have been developed to solve this wire delay problem [27], other problems such as ad-hoc hardware/software co-designs, functional validation of complex systems and the inconsistency between digital and analogue circuits still remain in SoC systems and are left for researchers and product developers to investigate their solutions.

Several SoC systems, integrated with separated processing cores to co-operate in increasingly complex and diverse tasks, promote the advent of new architectures. One popular new architecture is called Multi-Processor System on Chip (MP-SoC) [28] [29]. It greatly improves the system performance compared to a SoC system with a single processing core. Moreover, new communication architectures for efficient mutual interconnection of SoCs are investigated with increasingly massive SoC cores. Among all the potential solutions, Network-on-chip (NoC) [18], in which the chip-level communication interconnects tens to hundreds of sub-systems, is a very promising and effective architecture based on MP-SoC. It forms a subclass of distributed interconnection systems and a special type of MP-SoC, wherein each processor has independent functions and mutually interconnects a network instead of top-level wiring to improve the architecture, performance and modularity. It can process specific application tasks with heavy workloads or tight time budgets that a single processor may not be able to afford. It is a promising architecture for fulfilling demands on large data and mutually independent operations like audio/video processing and network protocol processing [27]. Many proposals and discussions of Network-on-Chip (NoC) have been made since the beginning of the 21st century [18] [19] [30] [31] [32]. As shown in Figure 2.2, a typical NoC involves SoC sub-systems that have their independent microprocessors, buffers/caches, buses and other function units. One such sub-system in NoCs, usually denoted as a Processing Element (PE), communicates with other PEs via routers through the whole on-chip network. The Network Interface (NI) in each PE, acts on as a bridge, conveys data between the local sub-system and the interconnection network.

NoC has regular and well-controlled topologies constructed from multiple point-to-point links by switches or routers that can reduce the design complexity [3]. A high-level parallelism of data processing has been developed in NoC systems such that one complicated big task is disassembled into several less-complex sub-tasks and deployed to different node PEs to be simultaneously processed, which considerably increases processing efficiency. Moreover, designers may also design the parallelism of data processing at other levels to improve system performance. Three levels of design parallelism are usually categorised: the bit-level parallelism, operation-level parallelism

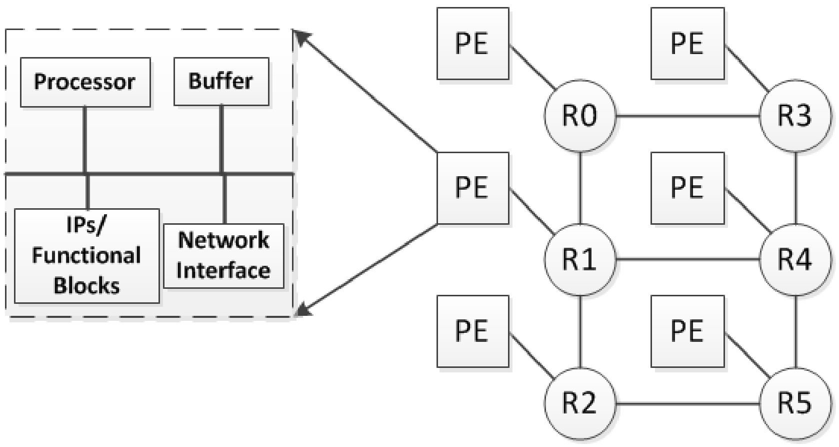


Figure 2.2: A Typical Architecture of NoC [3]

and task-level parallelism [27]. Among them, the task-level parallelism is most important to NoC architectures since it can be abstracted from a serial task and directly contributes to the performance improvement. Specifically, full usage of available network resources to implement proper task-level parallelism for complex and enormous data tasks can reduce design time cost. Besides, to the best of our knowledge, there is no a general tool that can properly abstract different levels of data-processing parallelism in NoC systems, leaving an open issue for further development of NoC modelling tools.

2.1.1.3 NoC Architectures

Modern NoC architectures basically consist of IP cores like microprocessors or Digital Signal Processors (DSP), on-chip memory and other functional blocks. Such PEs connected with routers, link the IP cores to neighbouring nodes to construct an NoC system [33]. Depending on different PE architectures, an NoC can be categorised into two classifications: *homogeneous* NoCs and *heterogeneous* NoCs. All the PEs in homogeneous NoCs have an identical internal structure, giving the same size of PE units and a symmetric chip, whereas the PEs in heterogeneous NoCs may have different unit structures. The size of its PE units is therefore diverse and the chip is asymmetric. Chip multi-processors may be homogeneous NoCs, as every PE is a processor [34]. Oppositely, MPSoCs may be designed to be heterogeneous because PEs may need different functions [35]. Each of these two kinds of NoC architecture has its merits and demerits [10]. A homogeneous NoC network has higher applicability and scalability for generic applications but less optimal performance for specific applications, while a heterogeneous NoC network performs optimal functions at the cost of prolonged design cycles for ad-hoc module designs.

According to a layer division proposed in [8], a network architecture can be divided into a *physical layer*, a *switching layer* and a *routing layer* from bottom to top. The **physical layer** describes how PEs are interconnected to each other. The **switching layer** decides how data messages are packaged and delivered, as well as how functional resources such as buffers, interfaces and links are allocated. The **routing layer** determines routing paths for data messages to traverse. In subsequent sections, major NoC design considerations and characteristics are stressed based on this layer division. In the physical layer, network topological structures are introduced. In the switching layer, flow control mechanisms will be mentioned. Different popular routing algorithms are finally elaborated in the routing layer.

2.1.2 NoC Topology

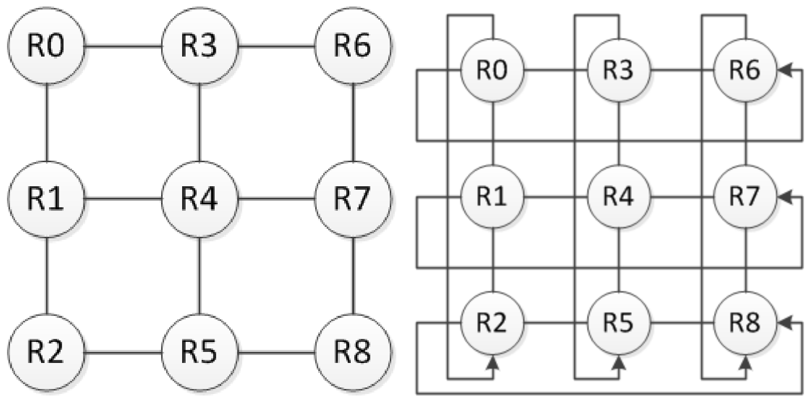
Network topology refers to the static allocation of channels and nodes in an interconnection network as the roads for packet transmission [3]. The NoC topology is one significant issue that substantially affects the layout strategies of the physical layer and performance of other layers. It is the fundamental step, also an essential step, of designing an NoC system.

If a node router connects to one or more PEs, such an NoC is a direct network like Mesh, Torus [36] and Tree [37] networks. Otherwise, it will be an indirect network if some node routers in the network are solely connected with other routers. Crossbar [7], Fat tree [9], Butterfly [38] and Butterfly Fat Tree (BFT) [5] network topologies are examples. There are also other special-use topological architectures like Nostrum, SPIN, CLICHE and Octagon, introduced in [39] and [6], that are application-specific NoC architectures.

2.1.2.1 Popular Direct Network Topologies

Mesh and Torus are the most popular topologies in direct on-chip networks [40]. The *mesh* topology is extensively used in recent research of NoC topologies. It is a two-dimensional orthogonal topology in which every node in the centre of a network has 4 neighbouring links, with 3 neighbouring links at border nodes and 2 neighbouring links at corner nodes. *Torus* is another commonly-used orthogonal topology and its structure is derived from the mesh topology. Its centre nodes have the same characteristics as in the mesh. The border and corner nodes have additional links to their opposite side nodes via wrap-around channels. This feature results in the increased interconnecting complexity and decreased power dissipation since the average side-to-side traversing distance of data packets is shortened. Classic topological structures of the Mesh and Torus are shown in Figure 2.3 [4].

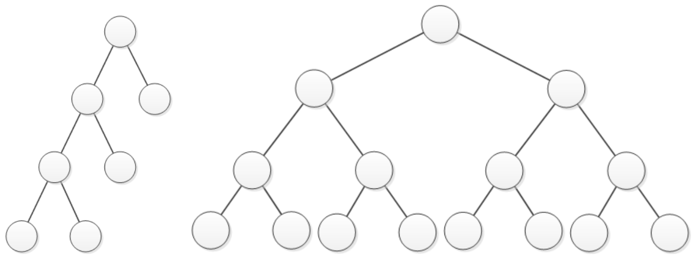
The *tree* topology [37] [9] is another popular direct network topology that has also been proposed in the last decade for minimising the network diameter with a given number of nodes and node degree. This network has a **root node** at the top (the **parent node**) that connects to a group of nodes (the **descendant nodes**). Each descendant node is



(a) Mesh Topology (b) Torus Topology

Figure 2.3: Mesh and Torus Topologies [4]

also connecting to a disjoint group (maybe empty) of descendant nodes. A node without descendant nodes is a **leaf node**. Hence in tree networks, one characteristic property is that every node has a single parent node except the root node, which means no cycles in trees. One network node can only reach another node at the same level as itself via their parent nodes (higher level nodes) instead of via their descendant nodes (lower level nodes). Additionally, if the distance from any leaf node to the root node is the same, that is, all the branches in the tree network have the same length, it is deemed a *balanced tree* topology. If the distances between leaf nodes and the root node are different, it is an *unbalanced tree*. There is also an indirect tree topology, *fat tree*, which will be introduced later. Figure 2.4 shows the balanced and unbalanced binary tree networks.



(a) Unbalanced Tree Topology (b) Balanced Tree Topology

Figure 2.4: Unbalanced and Balanced Binary Trees [5] [6]

For the NoC design, one major critical drawback for tree networks as general-purpose network platforms is that the root node and the nodes close to it may participate in most of the data transmission paths, which makes it become a severe transmission bottleneck especially when a lot of data is transmitted [5]. Once the paths near the root node are blocked or congested due to a large amount of data transmission, the overall network throughput will experience a dramatic drop and the traversing delays may become intolerably long. Besides, there are no alternative paths between any node pairs [6]. For a specific source-destination node pair, the link or connection of the node pair is fixed and unique. Therefore, if any segment of the traversal path is disconnected, a tree network will fail to transmit any data between the source and destination nodes. Such properties of unstable performance and low tolerance to disconnections make tree topologies not sufficiently adaptable to the constantly increasing data transmission requirements in modern chip designs.

2.1.2.2 Popular Indirect Network Topologies

Indirect network topologies have distinctive network architectures. Switches are designed to carry data communications between nodes, instead of the direct connections of nodes in direct network topologies. Such switch-based networks have evolved substantially over time so that a great number of topologies has been proposed, ranging from regular topologies for array processors and shared-memory UMA (Uniform Memory Access) multiprocessors to irregular topologies for Networks of Workstations (NoWs) [8]. In regular topologies, regular connection patterns are designed between network switches. While in irregular topologies, non-predefined patterns are built inside. According to the number of switches that data needs to traverse, both regular and irregular indirect networks can be further classified.

A *crossbar* network is one of the simplest indirect networks [7]. It enables a microprocessor in the network to connect to any other processor or memory unit so that multiple processors can be interconnected and communicate simultaneously with contention avoidance. Any new connection can be built up at any time if the requested ports/channels are available. It can be used for high-performance, small-scale, shared-memory multiprocessor designs; router designs of direct networks; and as basic components for large-scale indirect network designs. Figure 2.5 shows the architecture of an N by M crossbar network. The cost of a typical crossbar network is O(NM) [7], which will be prohibitively high if the numbers of N and M get large. Modern VLSI techniques only support definite numbers of pins on a single chip, which limits the size of large-scale crossbar networks integrated in a chip. For large-scale designs, partitioning one large crossbar network into smaller ones and implementing each smaller part on one chip is a feasible solution. Moreover, once two or more processors contend for the same memory access, an arbitration scheme will take effect to determine the proceeding order for each processor. The arbitration scheme can be less complex in this architecture than the one for a bus since conflicts in a crossbar are exceptional so that they are easier to resolve [8].

The *Fat Tree* [9] network is another indirect network. It is a multi-stage indirect network. Nodes in this network topology are only connected to the leaf nodes of the tree. As discussed, the root node in a tree topology and its close neighbouring nodes may easily

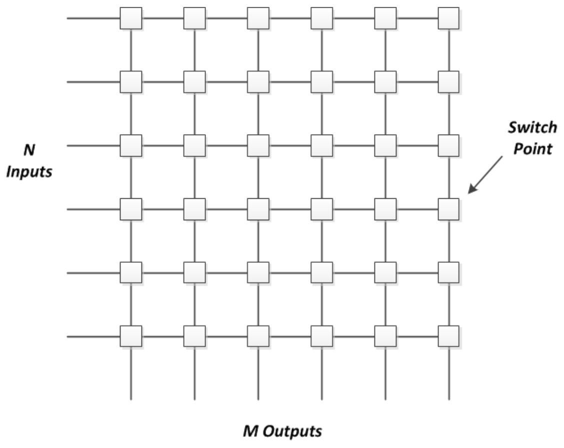


Figure 2.5: An N by M Crossbar Topology [7] [8]

experience high traffic load, which lowers the overall system throughput and causes traffic congestion. However, in the fat tree topology such a problem can be essentially alleviated by increasing the number of links at adjacent node switches near the root node. The closer those neighbouring nodes get to the root node, the more links are assigned to those nodes. This results in more channel bandwidth on those nodes that can tolerate higher traffic load. A simple fat tree topology is depicted in Figure 2.6.

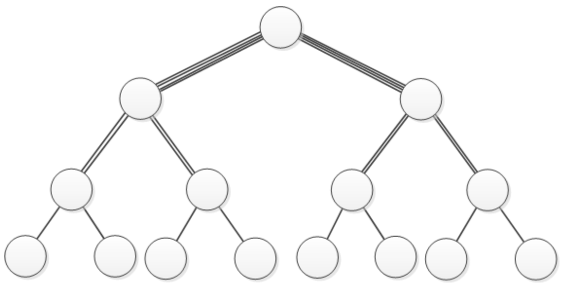


Figure 2.6: A Simple Fat Tree Architecture [9]

The *Butterfly* topology [38] is one popular multi-stage indirect network. As an example, Figure 2.7 shows a $2-ary$, $3-fly$ butterfly network in which the PE blocks of the left column have the same structures as the PE blocks of the right column. All the PE blocks in butterfly networks are connected via deterministic paths with equal transmission delays. The network between PEs in a butterfly topology can be replaced with other switching networks like crossbars. Thus, a $k-ary$, $n-fly$ butterfly network will consist of k^n nodes and n stages of k^{n-1} k *by* k crossbar switching networks. All of these switching networks offer fixed communication delays among all pairs of PE blocks. For such a network structure, one should also be aware that once contention occurs during data transmission, data may be blocked or even dropped.

The *Butterfly Fat Tree* (BFT) [5] network also belongs to indirect networks. It is a popular topology specifically designed for NoC applications. A typical BFT architecture is shown in Figure 2.8. In general, a pair of coordinates (l, p) is used to label every node

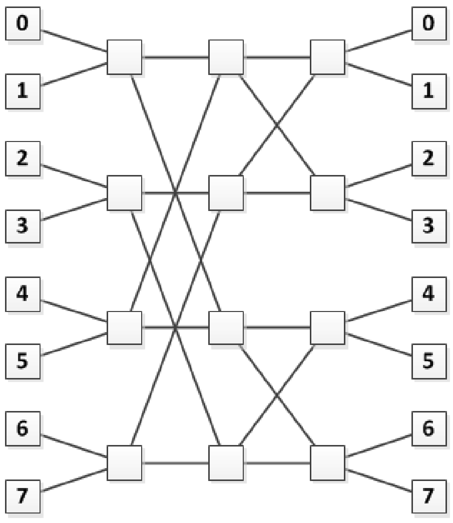


Figure 2.7: A $2-ary$, $3-fly$ Butterfly Architecture [10]

in BFT networks, where l denotes the level of a node and p denotes its position at the level. At the lowest level ($N = 0$ in the figure), N PE blocks are allocated, their addresses ranging from 0 to $N-1$ ($N = 64$ in the figure). The pair of coordinates $(0, N)$ denotes the locations of PE blocks at level 0. Each switch in BFT networks has four child ports and two parent ports, denoted as $S(l, p)$. The PE blocks will connect to $N/4$ switches at level $N = 1$. For N PE blocks, the number of levels will be $log_4 N$. Hence, there will be $N/2^j$ switches at the jth level. The number of switches in the BFT architecture converges to a constant, independent number of levels [5]. Consider the $4-ary$ tree as shown in Figure 2.8. The total number of switches at level 1 is 16 ($N/4$). At next levels, the numbers of required switches are reduced by a factor of 2 consecutively. The total number of required switches, S in the limit, will go to $N/2$.

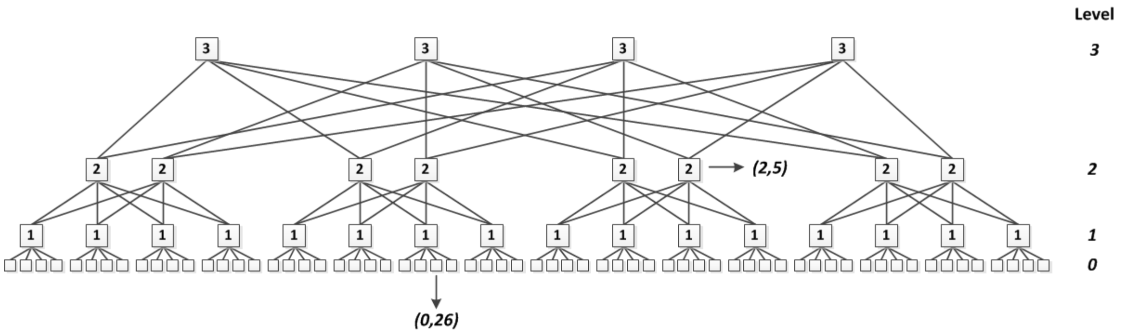


Figure 2.8: Butterfly Fat Tree Architecture with N=*64* PE Blocks [5]

Compared with direct networks, all the indirect networks can have equal minimal communication delays between all PE blocks. Yet the routers and channels of indirect networks need to be carefully calibrated to ensure equal delays and reasonable resource allocation [10]. Moreover, their network implementation is not as modularised as direct networks, which reveals high difficulties and complexity in designing properly applicable networks. In other words, the hardware implementation cost of indirect networks is normally higher than direct networks due to the more complicated network structures. In return, the performance of indirect networks may be more desirable for specific designs. Some particular irregular networks may offer perfectly matching features and ad-hoc performance for certain applications.

2.1.3 Flow Control

2.1.3.1 Basic Concepts

As per the theory in [8], the network communication could be deemed a service hierarchy starting from the physical layer that synchronises data transfers to higher level layers that perform diverse functions. It is helpful to distinguish operations of the NoC communication among three major layers: the physical layer, the switching layer, and the routing layer. Routing protocols that provide feasible routing properties are mostly implemented in the switching layer [11]. The switching techniques determine the ways internal switches are set to connect router port channels and the time when the data should be sent along the routing paths. Such switching techniques are tightly coupled with flow control mechanisms for synchronous inter-router data transmission and effective overall data forwarding, which is mainly introduced in this section.

Flow control is a synchronisation protocol for transceiving data [10]. It is closely coupled with buffer management techniques to decide how the router buffers work to respond to data requests, release their space, and decide how the blocked or waiting data is tackled in a network. In NoC designs, multiple communications are usually established concurrently and their routing paths are individually provided by the network. Contention will often occur as a result when more than one communication needs a single router or other network resource on their routing paths simultaneously. In this case, specific algorithms will be used to allocate available network resources for one specific communication, while others are forced to wait for further serving. The algorithms can allocate resources dynamically and considerably influence the router structure. Therefore, flow control methods are such algorithms to resolve resource allocation for contention.

A request/acknowledgment communicating mechanism is often used in flow control to ensure successful transfer and buffer space availability at the receiver. Commonly, the smallest data unit is divided to enable itself to be requested by the sender and acknowledged by the receiver. It is also worth clarifying the data levels delivered in on-chip networks, as used in this thesis. Flow control methods are normally needed at two levels. One is *message flow control* that occurs in packet-level data transfer. The other is *physical channel flow control* to forward a data unit through the physical link routers [8]. Switching techniques for the two levels of flow control differ due to the sizes of the physical and message flow control units. A complete data *message* need to be partitioned into several *data packets* for PE blocks of the network nodes to generate. Each data packet has a header containing destination node information and a payload that has the contents to be transferred. Yet a packet is still too large for physically routing in the network. So it can further be divided into *flits* (flow control digits [10], especially used in wormhole packet switching networks that mostly apply in this thesis), which can be transferred via physical link channels or switches at one time.

2.1.3.2 Switching Techniques

Flow control methods include two main categories: *circuit switching* and *packet switching* [10]. In *circuit switching* [41], a physical path from the source to destination nodes is allocated before data transmission. The initialisation of a routing path is implemented by injecting the header flit into the network which contains the destination address and some additional control information. When the header flit reaches the destination, a complete path is set up and an acknowledgment is sent back to the source. The message contents may now be transmitted under the full bandwidth constraint. The circuit of the routing path is released by the acknowledgement of the tail flit or the last few bits of the message reaching the destination.

As the routing path is set up in circuit switching networks, path service is guaranteed for the complete message transfer, which enables circuit switching techniques to be generally advantageous when data messages are infrequently transferred and request long transmission times compared to the path setup time. Yet the disadvantage is that once the path is set up, it will be reserved for the whole time duration of the message transfer and may block other messages that also request service. This substantially reduces the available data throughput of the network and the efficiency of the physical link usage. It may also cause circuit switching networks to suffer from intolerably long waiting times and significant power overheads for the path reservation especially when the network is experiencing heavy workloads [41]. Moreover, the base latency in circuit-switched message transfers is between the setup time of a routing path and the processing time the path needs for transmitting data. The transfer time of each flit from the source to destination nodes is based on the clock speed of the synchronous circuit or signalling speed of the asynchronous handshake lines [8]. The signalling or clock period need to be greater than the propagation delay of the routing path circuit. This implies that the speed limit of circuit switching networks may not function well as the system scales.

Apart from circuit switching, *packet switching* [42] transfers data immediately once generated by the PE blocks of network nodes. One message is partitioned and transmitted via several packets and routing paths for each packet are individually configured from the source to destination nodes. This dynamic routing configuration stores the data packets at each intermediate node before forwarding to the next node. Hence, it is also called *store-and-forward* (SAF) switching [43]. The header information of data packets is extracted by the intermediate node routers and used to arbitrate output directions for further forwarding.

Packet switching is advantageous if data messages are short and frequently transferred. Unlike circuit switching, where a partially reserved routing path may be idle for a significantly long time period, the physical links for communication between the source and destination nodes are fully utilised for data transmission in packet switching [8]. Several data packets for elaborating the same message may be transferred through the network concurrently even if the header packet has not yet reached the destination. Negatively, splitting one message into several packets for parallel transmission produces a data overhead for re-ordering at the destination node [11]. In addition, each packet must be routed at each intermediate node. If packets are routed adaptively through the network, packets from the same message may arrive at the destination out of order. In this case, the information sequence should also be contained in each packet header to correctly re-order the complete message at the destination.

Packet switching requires higher energy and more area than circuit switching since its data routing needs to be analysed dynamically and its switching configuration is more complicated [10]. Moreover, packet switching is more expensive because more buffer and storage area are used due to the extra information sequence contained in packet headers and the resource allocation contention of concurrently transmitted packets for one message at the intermediate nodes of a routing path. To solve the allocation contention, packet switching temporarily allocates network resources to one flit and reserves some buffer area for the blocked data. Depending on the buffer and storage reserved in each allocating process, packet switching networks can be divided into three types: a *store-and-forward* network [43] whose buffers are reserved in packet units, a *wormhole* network [44] whose buffers are reserved in flit units, and a *virtual cut-through* network [45] whose buffers are reserved in flit units but with sufficient buffer space ensured to store one whole packet.

Assume two packets arrive at two separate input ports and both request the same output port in each of these three types of packet switching networks. Contention in allocating the same resource (output port) to multiple data packets will occur. In the **store-and-forward network** [43], a packet is transmitted only if all flits have been received. Hence, the output port will be reserved for the earlier packet to send all its flits. The other packet has to be stored in buffers while waiting this reservation even if the output port is allocated but idle. This causes a long transmission delay and idleness of resources.

A **virtual cut-through network** [45] allows packets to be cut through to the input port of the next node router before the complete packet has been buffered at the current output port. So the output port of this type network starts transmission immediately once the header flit of the earlier packet is buffered. Due to the absence of blocking, the latency of header flits at each node consists of the routing latency and the propagation delay. The packet can be effectively pipelined through successive switches to reduce transmission delay. However, the buffer space cannot be reduced. Sufficient buffer space is required to allocate the complete packet in order to ensure the packet is stored in only one router instead of several if its header flit is blocked. In some cases, a flit can be exceedingly long and the area overhead used for storage will be intolerable if it is blocked.

In a **wormhole network** [44], packets are pipelined through the network and the buffer requirements within the routers can be substantially reduced compared with the virtual cut-through network, which keeps the required buffer space for one flit to the minimum. Buffers at a router are typically large enough to store a few flits of a partial packet. The complete packet is too large to be completely buffered within one router. Thus, at any instant time a blocked message occupies buffers in several routers. For two arriving packets requesting the same output, the output port in such networks is served for the earlier packet and only the head flit of the later packet is stored in the current router. Other flits of the later packet will be blocked in previous routers until the resources are allocated to serve them. This is the primary difference between wormhole and virtual cut-through networks. This reduced buffer space requirement for wormhole networks significantly reduces the average data transmission delay and enables the use of smaller and faster router structures.

2.1.3.3 Virtual Channels

Packet switching techniques assumed that (partial) data packets were buffered at port channels of network nodes. Buffers used in these cases are generally operated as FIFO queues. Thus, if one packet takes up a buffer for a port channel, no other packets can access it concurrently since the packet is blocked. Take the wormhole flow control method for example. Although it significantly reduces the required buffer space, however, the *head-of-line* (HOL) issue [46] may still introduce a long data transmission delay if the network experiences heavy workloads. A basic *head-of-line* issue is: When a data packet is blocked at an NoC node router, the router does not have enough buffer space to store the complete packet. As a result, parts of the blocked packet are stored in other routers that occupy their corresponding switches and port channels. These occupied resources then block other packets who request the same resources concurrently. In other words, the HOL issue is a deadlock issue in that a network state emerges where no packets can further advance because of the channel occupation contention of different packets.

To alleviate such an issue in wormhole networks, *virtual channels* (VCs) are introduced [47]. The basic idea is to share physical channels among multiple packets. When one packet is blocked, it will not occupy all the channels reserved for itself. Therefore, other packets can utilise the channels in the meantime. However, the unsent flits of the blocked packet cannot be dropped while other packets are using the same channel.

Extra buffers are needed and divided into several groups that connect to every port channel. Each such division is called a **virtual channel** (VCs) and is used to stores flits of one individual packet. One physical port channel can be partitioned into several VCs multiplexed across the physical channel. Each unidirectional VC is implemented by an independently managed pair of buffers as shown in Figure 2.9: Two unidirectional VCs in each direction cross the whole physical channel so that packets can share the physical channel at flit level concurrently, which can be considered as each VC uses a distinct physical channel operating at half the speed of the original channel.

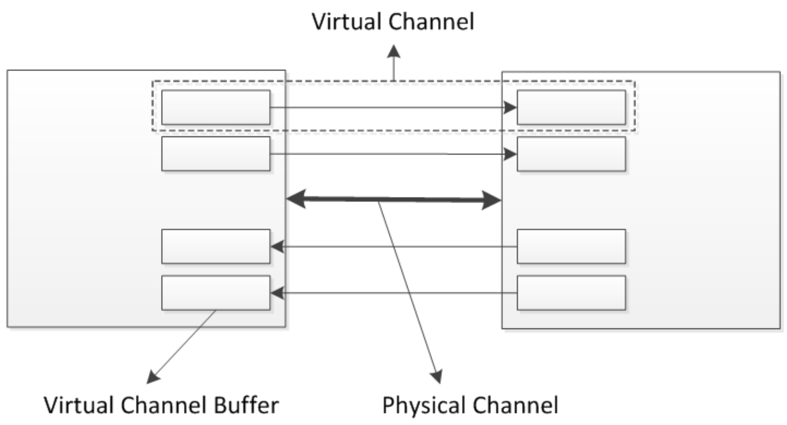


Figure 2.9: Virtual Channels [8]

Virtual channels can improve data packet latency and network throughput [4]. If packets are able to share a physical channel, they can be still in progress even when they are retained as blocked. The use of VCs decouples the physical channels, allowing multiple packets to share one physical channel concurrently in the same manner as multiple instructions shared one CPU (Central Processing Unit). The overall waiting time for a blocked packet to spend at a router for an allocated channel can be manifestly reduced, rendering an overall reduction in packet transmission latency.

Although each virtual channel can improve systematic performance by a small amount, the increased channel multiplexing will reduce the packet rate, which increases the latency. This increase may overwhelm the latency reduction due to blocking, resulting in an overall increase of average packet latency [8]. The increased number of VCs directly impacts on router and network performance. Besides, the VC arbitrator and allocator are also desirable to be integrated for better use of VCs, which complicates the router structure. Such structural complexity may cause further net increases in flow control latency of data transmission. Hence, the trade-off between efficient usage of virtual channels and packet transmission delay should be carefully balanced.

2.1.3.4 Quality of Service

A widely accepted concept of Quality of service (QoS) is the overall performance of a network, particularly the performance experienced by the users, namely the service the network provides to the users [10]. It is critically significant for data transmission under specific requirements. In the fields of distributed interconnection networks, QoS refers to the ability to offer different priorities for specific applications and data flows, or refers to the guarantee of certain performance to the data flows. It is crucially important especially when the network capacity is insufficient. To quantitatively assess the QoS, several related aspects of the network services are commonly considered, such as communication error rates, channel bandwidth, network throughput, transmission delay, resource availability and clock jitter.

Specifically in NoC systems, QoS is also an important factor. The network traffic communications usually have diverse requirements such as delay or throughput to fulfil, which is requested certain *guaranteed services (GS)* to manage the fulfilment for ensuring the network properly worked or functioned [25]. For instance, certain traffic communications may need operation targets to be attained before some deadlines such as the interrupt signals used in some real-time systems. The group of networks that support this type of service are generally deemed *delay-guaranteed* networks [8]. Circuit switching techniques can be used to implement **hard** delay-guaranteed networks in which various network traffic can always achieve the goals before deadlines. Other flow control methods such as Virtual Channels, are able to provide **soft** delay-guaranteed services in which the certain probability of traffic communications achieving the targets before deadlines can be guaranteed. Besides, many NoC traffic communications like multimedia can tolerate particular delay variance but are sensitive to overall throughput. The networks that satisfy this requirement are called *throughput-guaranteed* networks [25]. Nearly all modern popular flow control methods are able to provide such services by assigning high priorities to the throughput impact factor over others to the data transmission.

Most network traffic communications do not have extra specific performance requirements when a network is fast enough. It is always beneficial to transmit data as soon as possible but the failure of data transmission should not break the whole network. These traffic communications are called best-effort (BE) traffic [25]. Both wormhole and virtual channel flow control techniques can provide this service. However, an alterative concept to the service is also adopted, which regards it not as a type of QoS but as an alternative to complex QoS control mechanisms. By offering the BE service, a network can provide high-quality traffic communications by over-provisioning the capacity. Hence, it has sufficient resource allocation for the expected peak traffic workload, which should result in the absence of network congestion and finally eliminate this need for QoS control.

2.1.4 Routing Algorithms

2.1.4.1 Basic Concepts

In the routing layer of NoC architectures, routing algorithms essentially impact on the system performance by determining the routing path through which a data packet traverses a network of a certain topology [10]. There are many routing algorithms designed for fulfilling various specific and distinctive requirements. However, if a routing algorithm is not adaptive, either overloaded resource use or idle resource waste or even both may cause network congestion. This adaptivity problem is complex and difficult so that no general routing algorithm can perfectly fit any type of traffic communication. Customising specific routing algorithms to meet various design specifications is necessary to obtain desirable performance.

Moreover, it is noteworthy that although distinctive routing algorithms that fulfil certain requirements may not be adaptive, they can be used in some other traffic communications with totally different specifications [3]. In other words, one specific routing algorithm may be suitable for many communication cases. But only in a few of these cases it could provide optimal or desirable performance. For this reason, comprehending a representative set of routing algorithms proposed for typical modern NoCs is helpful to categorise the spectrum of existing algorithms, and facilitating designs of suitable algorithms to various network topologies. Many properties of NoC networks are a direct consequence of their adopted routing algorithms, the parts of which that are of particular concern in contemporary industrial chip designs are listed below [48] [4]:

- **Connectivity**. This property is the ability of the algorithm to route data packets from the source node to the destination node in a network.
- **Adaptivity**. This property is the ability of the algorithm to route data packets via alternative traversal paths if contentions or malfunctioning components occur in a network, that results in a block or disconnection at specific traversal paths.
- **Deadlock and livelock freedom**. This property is the ability of the algorithm to guarantee that data packets will not block or wander through the network traversal path ceaselessly in a network. These issues are explained in next section.
- **Fault tolerance**. This property is the ability of the algorithm to route data packets when faulty, malfunctioning components or unexpected disconnections occur in a network. The fault tolerance of an algorithm seems like to imply its adaptivity. Yet it can also be achieved without adaptivity by routing a packet in multiple phases and storing in some intermediate nodes. Extra hardware may be needed in the network for implementing this property.

A routing algorithm that is adaptive to one special traffic communication over a network may not fit the same network with alternative traffic patterns [25]. More specific, the traffic communication in real-world networks may not always be predictable or foreseen at the early design stages. It could vary dynamically during run-time operations so that a currently adaptive routing algorithm may no longer be adaptive at another time. Furthermore, if a fault or malfunctioning components occur in a network, the network topology would be changed temporarily or permanently [8]. This unintended topological change may make the adopted routing algorithm no longer adaptive, deadlock-free or feasible. Consequently, both the adaptivity to design requirements and the effectiveness during the whole operation process should be considered to design a proper routing algorithm for a specific application.

2.1.4.2 Deadlock, Livelock and Starvation

Intermediate nodes may be unavoidable for data routing across the whole network before arriving at the destination. Specifically in switch-based networks, the data will also reach the destination node by routing through intermediate nodes. In practice, failure of arrivals in data packet routings may occur even if the network supplies fault-free routing paths. Given a routing algorithm that is capable of utilising the supplied routing paths in networks, there still exist certain reasons that may prevent successful data traversal.

Deadlock is one typical reason. It may occur in a network when packets are no longer able to make further progress because they are waiting for one another to release the resources held or blocked by themselves, which are usually buffers or channels [3]. If a sequence of such waiting packets forms a closed cycle, the network will be deadlocked. Since buffer capacity is finite in networks, a deadlocked situation has a chance to happen when each packet whose header is on its way to the destination requests buffers of intermediate nodes who are keeping themselves fully storing packets so that not a single packet can advance toward its destination. All the packets involved in the ceaselessly waiting cycle are blocked forever, which stops the data transmission and paralyses network operation. Figure 2.10 demonstrates a simple yet classic example of network deadlock [11]. A grid network consisting of four nodes is given. At each node there is a data packet that is ready to route forward to the subsequent node in a clockwise direction. However, these four packets can never make forward progress since they are waiting for the network links other packets are occupying.

Livelock is another reason that may cause the network transmission failure. It is related to the network pathology of deadlock. In livelock, data packets continuously route across the network but are not able to arrive at their destination even if they are not blocked permanently at intermediate nodes. Livelocked data packets may occur because the destination channels they request are held by other packets or even a faulty routing decision has been made and kept. Livelock commonly occurs in non-minimal

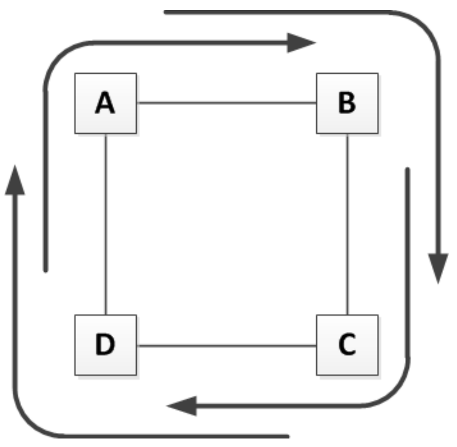


Figure 2.10: Example of a Typical Network Deadlock [3]

data routings since data packets may be easily misrouted away from their destination. Strictly limited misrouting times to the data packets in non-minimal routing networks are necessarily guaranteed [3]. Figure 2.11 explains one example of livelock [10]. In a close cycle four-node grid network, one data packet is continuously circulating around the network targeting a nonexistent destination node $PE(4)$. This fault may possibly occur in a transient way, such as the drop of a packet header due to unexpected fault attacks or a permanent network disconnection.

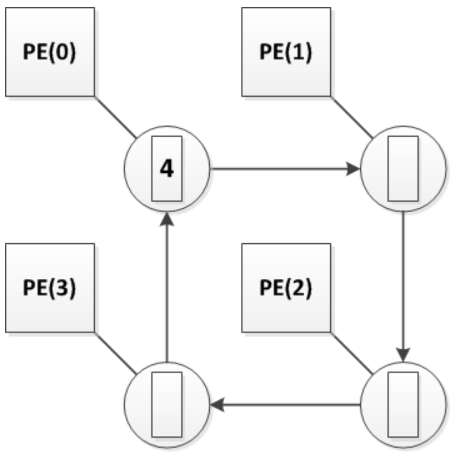


Figure 2.11: Example of a Typical Network Livelock [10]

Starvation is another potential reason for network communication failure. Starvation is a situation in which a data packet is permanently stopped from forwarding due to intense network traffic or constant resource allocation to other packets who are requesting the same resource in the meantime with the stopped packet. Starvation usually occurs when an faulty resource allocation is continuously used in arbitrating the conflict of resource requests [8].

The occurrence of deadlocks, livelocks, and starvation is attributed to the finite number of available resources in networks [8]. For NoC designs, it is extremely significant to remove deadlocks, livelocks, and starvation, or at least to be able to recover from these issues. In other words, either avoidance, prevention or recovery schemes are necessary no matter what routing algorithm is used in NoCs.

Starvation is relatively easier to resolve compared with the other two. Since faulty resource allocation is the root of starvation, guaranteeing a correct resource allocation scheme should be a feasible solution [3]. For instance, when different priorities to packets are used, a proper starvation prevention would be reserving certain channel bandwidth for low-priority packet routings. This can be achieved by the throughput limit of high-priority packets or assured reservation of virtual channels or buffers for low-priority packets.

The easiest way of avoiding livelock in NoC design is using minimal routings only [11]. Using minimal routings instead of non-minimal ones could improve the throughput performance especially in wormhole switching networks as the data packets will no longer consume extra network resources other than strictly necessary ones for their source-destination routings. To introduce fault tolerance to the network is the essential motivation of using non-minimal routings. In the cases where high fault tolerance is necessarily demanded, so that non-minimal routings have been used, strictly limiting the numbers of misrouting operations to data packets is also effective to prevent livelock.

Deadlock may be the most difficult to solve. Three strategies may be derived for deadlock handling: deadlock prevention, deadlock avoidance, and deadlock recovery [49]. In *deadlock prevention*, network resources such as communication channels or router buffers can be completely reserved before the data transmission starts if they are needed by data packets [8]. Such a resource reservation scheme helps prevent the resource requests of the packets that cause a deadlock. In *deadlock avoidance*, network resources are successfully allocated to the requested data packets that advance along their routing path only if the resulting network state is safe. This strategy should avoid the update of network state caused by sending additional packets. Moreover, deadlock can be avoided by eliminating cycles in the resource dependence graph [3]. Specifically, deadlock freedom can be guaranteed if routing cycles generated in the routing algorithm are avoided, or the buffer occupied in a cyclic manner in the flow control protocol. In *deadlock recovery*, a detection mechanism becomes a must if network resources are allocated without checks. Once a deadlock is detected, a deadlock recovery mechanism is necessary to enable the deallocation of network resources from original requests and granted to others. Packet abortion at occupied network resources or even a temporary halt of the overall network may be demanded in order to re-allocate resources.

Deadlocks, livelocks and starvation paralyse a part or even the whole network [10]. A practical NoC system must either ensure that its routing algorithm is deadlock-free or livelock-free, or supply effective recovery schemes if deadlock or livelock occur. Adaptive routing algorithms may be prone to produce deadlock because they usually have routing cycles in the extended channel dependency graph [50]. Such unintended routing cycles can be eliminated by restricting the turn model [51] or adding extra virtual channels [52]. Note that even a deadlock-free routing algorithm may still produce deadlocks if permanent change of network topology occurs and fault-tolerant adaptation is missing. Instead of deadlock-avoidance design, a deadlock recovery scheme should be concerned, though such schemes may degrade network throughput performance significantly [53].

2.1.4.3 Typical Classification of Routing Algorithms

In the last decade, many deadlock-free routing algorithms have been proposed for multiprocessing systems [51] [12] [54]. Some of them are designed for the NoC domain like the Odd-Even [12] and DyAD [55] routing algorithms. Others have inspired the design of new routing algorithms for NoC systems [56]. For instance, the turn-prohibition algorithm [57] has been proposed based upon the turn model [51]. Routing algorithms for NoC systems can be classified in many ways. In general, the classification can be determined according to 3 different criteria [48]:

1. **how a routing path is defined:** *deterministic* and *non-deterministic* routings can be classified as per this criterion. A single routing path is completely supplied between the source and destination in deterministic routings while in non-deterministic routings, the routing path is a function of the instantaneous network traffic [58]. The adaptivity of non-deterministic routing algorithms is supported by practical network status, so they are also considered as *adaptive* routing algorithms. Non-deterministic routings supply multiple possible routing paths that can be used by a packet to reach its destination. However, deadlock and livelock may occur when the full adaptivity is provided by the algorithm [59].

2. **where the routing decisions are taken:** *source* and *distributed* routings are classified by this criterion. Source routings determine the whole routing path at the source node prior to the data transmission. The calculated routing path is stored completely in the packet header [60], which may cause packet size overhead. While in distributed routings, a distributed manner is used to determine the packet route direction at each intermediate node. Since each next hop direction is made based on local network knowledge, the packet header could be very compact. Distributed routings may be very cheap and simple to implement in regular network topologies because one routing decision is applicable to all the network nodes. Besides, distributed routings can also detect and avoid faulty or disconnected paths, resulting in increased fault tolerance.

3. **the length of the routing path:** routing algorithms can be categorised as *minimal* or *non-minimal* routings [59] [60] due to this criterion. Minimal routing algorithms only supply routing paths that carry the data packets closer to the

destination node, which guarantees the possibly shortest routing paths. In non-minimal routings, routing paths that may take the data packets away from their destination can be supplied, which offers higher flexibility that allows any available routing path between the source and destination nodes. This is practical when fault tolerant routings are desired, since such routings support alternative routing paths even when all possible minimal routing paths are faulty or disconnected. However, non-minimal routing algorithms may suffer from livelock issues and usually cost more network resources.

Table 2.1 [4] lists the popular routing algorithms currently proposed for NoC systems according to the discussed classification. The detailed formation and properties of these routing algorithm will be introduced in the subsequent sections.

Table 2.1: Overview of Popular Routing Algorithms in NoC Design [4]

Routing Algorithm	Path Definition	Routing Decision	Path Length
XY/YX	Deterministic	Distributed	Minimal
OE	Adaptive	Distributed	Minimal
Source	Deterministic	Source	Non-Minimal
Q	Adaptive	Distributed	Non-Minimal
MaXY	Adaptive	Distributed	Minimal
PROM	Oblivious	Distributed	Minimal
DyAD	Adaptive	Distributed	Non-Minimal

2.1.4.4 Deterministic and Adaptive Routing Algorithms

The evident advantage of deterministic routing algorithms is their simplicity of hardware implementation due to less-complex routing module requirements. They are also suitable for routing in irregular networks where non-deterministic routings may not work. However, the lack of flexibility in selecting proper routing paths in deterministic routings easily causes network congestion especially when heavy workloads are undertaken and multiple communications request the same resources at the same time. Moreover, when the network topology is slightly changed due to faulty components, deadlock issues may occur if deterministic routing is used [61]. Thus, using non-deterministic routing algorithms is better in networks with high demands for fault tolerance.

A special kind of routing algorithm called dimension order routing (DOR) [52] [8] are extensively used in regular NoC topologies. They do not carry information of the whole routing path but only the destination address. Data packets are routed via the rectangle edges of such topologies between the source and destination. Consider the distance between the current node and destination node calculated as the sum of offsets in all network dimensions. DOR algorithms will route data across dimensions in strictly increasing or decreasing order. The distance offset in one particular dimension will be reduced to zero firstly, then routing data at another dimension to reduce its dimensional distance offset subsequently. DOR algorithms will produce deadlock-free algorithms when used for n-dimensional mesh networks. Specifically in 2D-mesh, they are well known as XY routing algorithms. Many modified NoC routing algorithms are directly derived from XY algorithm for various specific designs. In Figure 2.12, an X-first XY routing algorithm is shown through the four different routing examples [11].

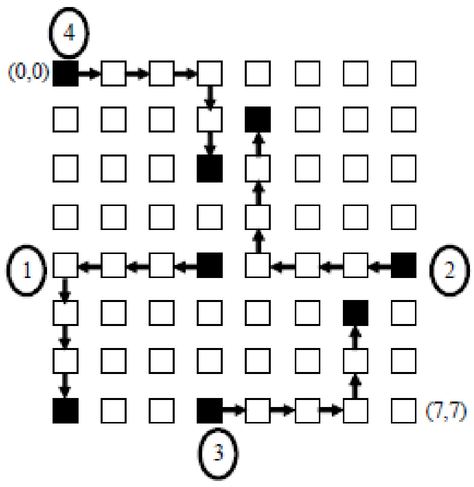


Figure 2.12: XY Routing Algorithm for 2D-Mesh - X First [11]

DOR algorithms are regarded as source routings that can achieve an extremely low area overhead. They are also considered as distributed routings since the routing decisions can be made by routers at intermediate nodes in the routing path. When the network is not experiencing heavily traffic load, DOR algorithms can always ensure minimal routing paths, leading to the best transmission delay performance. Due to these advantages, DOR algorithms have been used in many NoC designs [62] [63] [64].

A major disadvantage of deterministic routing algorithms is the lack of path flexibility that may lead to network congestion and worse data transmission delay when network traffic is under a heavy workload. Non-deterministic or adaptive routing algorithms, however, can overcome this disadvantage by routing data along alternative paths and distribute the traffic to all network resources based on network states. Algorithms with no restrictions are called *fully adaptive*, otherwise they are called *partially adaptive*. Fully adaptive routing algorithms are subject to deadlock conditions [48].

Different adaptive algorithms forward data packets driven by various motivations. Glass and Ni in [51] have proposed a turn model, as shown in Figure 2.13, for designing wormhole routing algorithms for mesh and hypercube network topologies with deadlock and livelock freedom by prohibiting certain turns to avoid the formation of any cycle.

The Odd-Even (OE) algorithm proposed by Chiu in [12] and the Dynamic Adaptive Deterministic switching algorithm (DyAD) [55] attempt to balance the traffic load in the network and improve overall throughput. The OE model avoids deadlock by restricting the locations where certain turns can be taken. The determination of its routing path is governed by the following rules: for any data packet routed by OE algorithm, the following turns are not allowed (column starting from 0):

1. East-to-North or East-to-South at even columns.
2. North-to-West or South-to-West at odd columns.

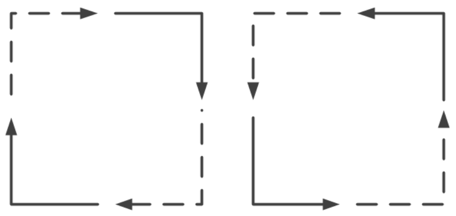


Figure 2.13: Example Turn Model for Adaptive Routing [12]

Figure 2.14 illustrates the examples of allowable data routings in a 9 *by* 9 2D-mesh network using the OE routing algorithm [3]. In the figure, East-to-North turns at even columns and North-to-West turn at odd columns are not forbidden. Four different routings in which S and D denote the source and destination nodes.

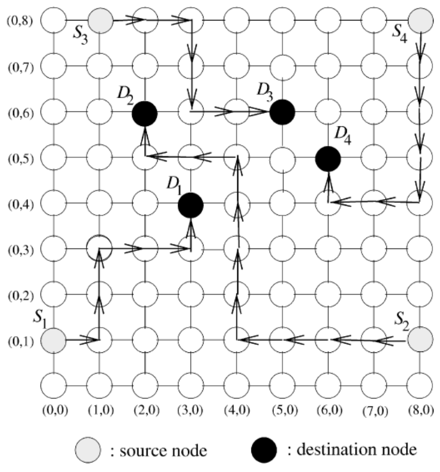


Figure 2.14: Examples of OE Routing in 9 by 9 2D-Mesh [3]

Some other algorithms like [65] and [66] have a fault tolerance property by re-transmitting data packets when a broken communication link is detected. Yet it is usually expensive to apply such adaptive algorithms since considerable area, energy or transmission delay overheads may be needed for gathering precise desirable network states like network congestion and faulty nodes.

2.1.4.5 Source and Distributed Routing Algorithms

Most deterministic routing algorithms are source routing based [10]. The data sender at the source node attains the knowledge of destination address and thus specifies the routing path in advance. The selected routing path is usually recorded in the packet header to implement source routing. Each intermediate node on the selected routing path analyses the heading information carried by the packet header, firstly deciding the output direction, then removing the turn from the recorded information, and eventually forwarding the modified packet header to the next node.

Source routing has been used in many NoC designs [67] [68] [69] [34] since it can be used in both regular and irregular topologies with very low hardware overheads. However, the complete routing path is recorded in the packet header. This may cause substantial size overheads especially when the routing path is extremely long and a large amount of routing information is recorded. In this case, extra transmission delay and energy consumption will result.

Q-routing [70] is a distributed routing algorithm based on machine learning. With this algorithm, each router at the intermediate nodes of the routing path acts as an agent that collects local information and routes along the direction in which the minimal real-time routing cost from the current node to the destination node can be retained. Each router of the intermediate nodes maintains the estimated routing cost of a packet to its destination via each neighbouring direction of the node. When a router sends a packet to one of its neighbouring nodes, the neighbouring node sends its estimated cost back. With this real-time local information, the router updates its estimated cost and arbitrates the most cost-efficient routing direction. This property helps to adapt to congestion through an updated real-time direction that has the least estimated cost. The overall routing path may not be minimal.

2.1.4.6 Minimal and Non-minimal Routing Algorithms

One typical **minimal** routing algorithm is the QoS-aware Minimally Adaptive XY routing algorithm (MaXY) [71]. MaXY is a variation of the XY routing algorithm that retains the advantage of XY routing simplicity, and also has the capability of adapting to the network traffic. MaXY algorithm ensures that minimal routing path and livelock freedom are always given by determining the next routing direction only from the 2 path-length reducing directions at any stage. Besides, the algorithm is adaptive and decisions on hop directions are made at crucial node positions.

In the MaXY algorithm, data routing aims to equalise the absolute difference between the distance offsets of the current and destination nodes along the X and Y dimensions. So firstly data will be routed along the dimension which helps to reduce the absolute difference of distance offsets between the X and Y dimensions [71]. Once they are equal, packets are routed along the dimension which has more buffer space. If the paths have the same buffer space along both dimensions, then a random selection of routing dimension is made. It is clear that the next node of the routing path in MaXY algorithm is always closer to the destination node than the current node, which indicates a minimal routing path is adopted while also having a certain adaptivity to distribute the traffic load [25].

Oblivious routings can be either **minimal** or **non-minimal**, but should be distinguished from deterministic routing although they are considered to be identical in some cases [8]. Generally speaking, routing decisions made by oblivious routing may be independent of, namely oblivious to, the network states. Yet, it does not necessarily mean that such routing decisions are deterministic. For example, at one intermediate node on the routing path, the next routing direction may be determined among several candidate choices based on some certain fashions that are irrelevant to network states. In this case, a deterministic routing will always make the same specific choice from those candidates every time. But oblivious routing may choose alternative choices from the candidates at different times.

Oblivious routing algorithms route data without regarding the states of the network [18]. Two Valiant's randomised routing algorithm [72] is a **non-minimal** routing that can balance the traffic load in nearly any network topology. An intermediate node is randomly selected for each data packet to relay and then forwarded to the destination node. The stochastic routing algorithm [73] [74] is another oblivious **non-minimal** routing algorithm in which a data packet is forwarded to multiple output ports in each intermediate node of the routing path. This results in flooded data from the source to the whole network. This algorithm has strong fault tolerance and guarantees the successful delivery of a data packet to the destination as long as the destination node is reachable, though such a data flood may severely waste network resources and cause extremely low overall network throughput.

One oblivious and **minimal** routing algorithm used in NoC design is the Path-Based, Randomised, Oblivious, Minimal Routing algorithm (PROM) [13]. It is suitable for NoC applications with n *by* n mesh geometry and is deadlock-free when the network routers have at least two virtual channels. The PROM makes the same decision progressively via efficient, locally randomised directional arbitrations at each intermediate node of the routing path. The routing decision of the PROM algorithm is made randomly and only for the next hop conforming to the minimal-path constraint. A PROM example is given in Figure 2.15 to illustrate the random choice scheme of the PROM algorithm. At each intermediate node, the next hop is chosen by a fair coin toss. Note that each hop choice in PROM routing is oblivious and computed locally [4]. This property may not be guaranteed to obtain the optimal routing performance but to give a competitive network throughput under various traffic situations. The packet header used in the algorithm could be compacted without a size overhead.

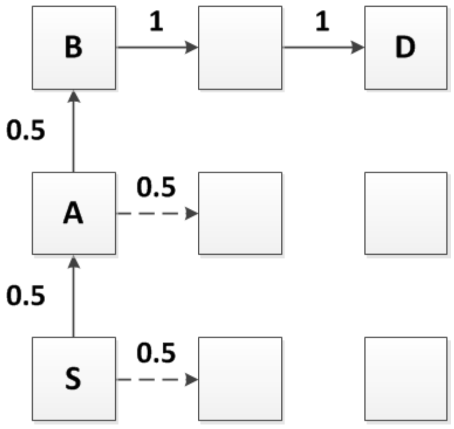


Figure 2.15: PROM Example: Randomly Choose the Minimal Routing Path [13]

Hu and Marculescu in [55] proposed the Dynamic Adaptive Deterministic switching algorithm (DyAD), which combines the advantages of both deterministic and adaptive routing algorithms in order to better-distribute traffic load and improve the network throughput performance. The routing algorithm works in a deterministic mode when the network load is not heavy enough to cause network congestion. It judiciously switches to adaptive mode when the network becomes congested. The DyAD algorithm is a typical routing algorithm that supplies **non-minimal** routing decisions [4]. Providing a minimal routing path between the source and destination is not the most significant concern in this algorithm. The purpose of designing DyAD is to propose a routing scheme that balances the traffic load with overall network throughput performance. Hence, using alterative, non-minimal routing paths to route data away from the destination node is allowed in DyAD when all possible minimal routing paths are congested.

2.1.5 Commercial Employment

Since NoC networks have a number of advantages in performance optimisation, design area and power consumption compared to the traditional bus-based SoC systems, commercial employment of NoC products is attractive to SoC makers and chip-design industries. Two main companies, Arteris [75] and Sonics [76], provide industrial NoC products and IP interconnection solutions to semiconductor products in many fields. Other companies also propose multicore chips for different uses.

Arteris [75] is an NoC company that addresses increasing design and performance complexity of modern SoC industrial products by providing Network-on-Chip Interconnect IPs for any System-on-Chip. It claims to have pioneered the first commercial NoC solution for over 50 tapeouts of SoC products. Its patented NoC interconnect IPs provide flexible and scalable solutions that allow designers to optimise and achieve the specific design goals like improved performance, reduced timing closure, smaller area or higher scalability for their particular products. Its main NoC products contribute to IPs for SoCs with different functions such as faster timing closure, high performance, critical mission and small area. These Arteris NoC solutions are extensively integrated in many SoC systems for new ultra-HD Televisions, servers for next-generation cloud data centres, high-end consumer electronics, portable Internet multimedia terminals, ultra-low power wireless connectivity for the Internet of Things (IoTs), G.fast-based modems and so on.

Sonics [76] is another company selling on-chip network products. As per its claim, Sonics has produced the industry's first GHz network-on-chip. Other its current products include complex multi-core cloud-scale SoCs, industry's highest frequency NoC, consumer-friendly multi-subsystem SoC infrastructure, configurable system IP blocks and low-latency communication NoC. Similar to Arteris, Sonics also cooperates with a number of other semiconductor and consumer electronic companies to integrate its own NoC products to a wide range of other SoC products. Their collaborative system products include new application processors for smart appliances with high speeds, secure sound and image communications, integrates NoC IP cores for advanced satellite modem SoC, integrated customer-specific application processors for EDA tool and manufacturing flows, NoC IP for heterogeneous multi-core devices and so on.

Besides, other semiconductor companies have also launched their NoC processors for commercial and research uses. IBM CoreConnect architecture [77] was proposed in the beginning of the last decade. It enables the integration and reuse of up to 8 synchronous master processors for standard and custom SoC designs. The Intel Teraflops research processor chip (also called Polaris) [34] is another NoC product launched in 2007. It connects 80 simple cores by using a tiny, non-IA (Intel Information architecture) instruction set [78]. This multi-core processor is the first chip developed by Intel Corporation's TeraScale Computing Research Program to explore tiled, 2D-mesh architectures for tackling cache coherence in scalable, next-generation processors. The second NoC product of Intel TeraScale Research Program is the Intel Single-Chip Cloud (SCC) Computer [79] launched three years after the Teraflops chip. SCC uses a mainstream x86 instruction set to connect 48 Pentium class IA-32 cores [80] on a 6 *by* 4 2D-mesh network. It explores the scalability of many-core architectures for not using a cache-coherent shared address space. TILE-Gx line of processors [81] are NoC products of a multicore processor family developed by Tilera Corporation (which was acquired by EZchip in 2014) [82]. The TILE-Gx multicore processors consist of a mesh network connecting up to 100 processing cores, which provide high-performance network processing for a wide range of applications.

2.2 System-level Design Automation

2.2.1 Current Opinions

Due to the increased demands on new design techniques in modern chip industries, high-level design automation is needed to emulate behaviours and performance of highly-integrated embedded systems [32]. The design trends have gradually moved from RTL to ESL (Electronic System Level), abstracting high-level models in terms of architectural design, algorithm development, hardware/software partitioning and co-design, prototype establishment and feasibility verification to specific architectures [27].

As per the high integration of NoC architectures, new design methodology and simulation tools are desirable for global system design and performance analysis in order to shorten product life cycles and fulfil customisations [83]. In this case, high-level abstraction models and system performance estimations based on fast and accurate hardware/software co-simulation platforms are considered as one possible solution [84]. Yet this solution also brings a new design challenge, which is the difficulty in developing accurate and efficient behavioural models and performance estimates without low-level implementation and long experimental time. To meet the challenge and hence improve the productivity of NoC designs, development of advanced design automation techniques are necessary to abstract accurate-enough system-level architectural models, integrate application-specific modules, and optimise task mappings for limited resources.

A typical system-level design flow that is widely accepted by modern chip developers is the Y-chart process shown in Figure 2.16 and advocated by [14]. With custom specifications the required applications and available platform resources are determined. Then those applications are mapped and scheduled onto the systems modelled by the supplied resources. Relevant performance metrics are evaluated based on the generated mapping schedule. The design space for specific functions are also explored in this stage. After this, the system architecture will be implemented and iteratively refined until design specifications are entirely fulfilled [20]. More design space exploration will also be developed to determine low-level design specifications. Finally, a set of executable tasks and system architecture will be abstracted for the subsequent design stages.

In all system-level design characteristics, **performance estimation** is usually a significant factor that needs to be carefully considered [83]. Many design decisions need to be particularly made to meet special design specifications in order to avoid unsatisfactory system functions. Moreover, it can be very resource-costly and time-consuming to iteratively refine complex system designs if the high-level models are inaccurate and inefficient. As a result, the *accurate model abstraction and performance estimation* of system-level design automation is requisite to reduce design errors and shorten design time cycles, which increases the design productivity. Correspondingly, *critical designs in simulation models* would focus on the ability to capture real workload

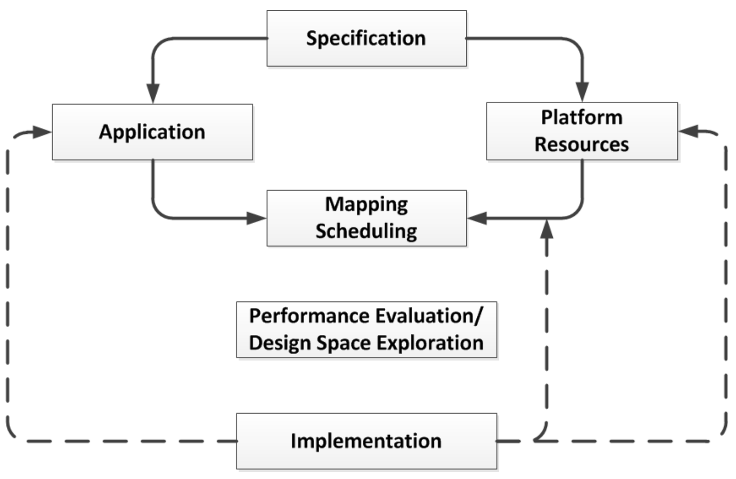


Figure 2.16: Typical High-Level System Design Process [14]

scenarios of applications [14]. Applying a *time-based simulation methodology* is necessary and effective to gain the dynamic simulation statistics such as instruction execution delays and interconnection delays. Once achieved, the system architecture and its accurate performance estimation can be iteratively refined based upon those results. *Fast simulation* is another important factor for high-level performance estimates since it is helpful to mitigate time-to-market pressure and facilitate the further design stages. Due to the conflict between high time-to-market pressure and large time cost for capturing realistic and accurate performance, developing more precise high-level system models and performance evaluation techniques may be a potential solution to weaken the conflict.

For digital system design simulations, *Instruction Set Simulation* (ISS) is used for software design of digital electronic systems to obtain the execution impact and real-time simulation data by its support of cycle-accurate simulations [27]. Yet this kind of simulator has the disadvantages of low simulation speed and high model complexity, which is unaffordable for system-level design automation at the early design stages. Moreover, ISS provides excessive and dispensable details for high-level model abstraction. The simulation speed and model complexity are sacrificed at the cost of accuracy improvement, which cannot fulfil the accuracy demand on system-level NoC designs.

Therefore, to meet the demands on modern early-stage design automation for high-level design flexibility, a simulation method based on *native software execution* to achieve system performance evaluations has been proposed and widely accepted [85]. A native simulation methodology indicates that the simulation models compile and execute each program of the design directly on a host machine instead of the target processor used in ISS. Several issues need to be dealt with to develop such models, such as timing, module function interpretations and coupling of function interpretations with performance models. For timing analysis, the native simulation method would execute binary code on the target processor with performance models to get important and accurate timing effects. The choices of current native-based simulation methods for functional interpretation level classify their techniques. Usually, their functional interpretations would be **binary level**, **intermediate representation level** and **source level**. The coupling of function interpretation and performance models can be realised by annotating interpretation with timing information [14].

At different functional interpretation levels, *Binary Level Simulation* (BLS) translates binary codes while compiling to gain functional interpretation that could provide faster simulation speed than ISS because the fetching and decoding of time-consuming instructions are implemented prior to simulation. *Source Level Simulation* (SLS) interprets timing analysis of system like software execution delays on the target processor by source code following the mapping of source code and machine code [14], which leads to high simulation speed and low design complexity that is suited to high-level designs. Yet one of its shortcomings is its low accuracy of estimation results due to the trade-off on speed improvement. Besides, SLS cannot precisely estimate optimally-compiled codes as the accurate timing statistics depend on mapping between source and binary codes, which may be destroyed by optimisations of the compiler [27]. While most current system designs need to compile their programs with a compiler, this disadvantage will strongly hinder the adoption of SLS. Thus, a middle-level methodology using intermediate representation level techniques is proposed in order to achieve accurate timing analysis that transforms source code of programs at lower levels, such that its structure will be close to binary code [84]. The current simulators for distributed interconnection systems need to improve their models.

Generally, system performance estimation can be divided into two groups: *static* and *dynamic* simulation [14]. Dynamic simulation can capture real workload scenarios of system design to estimate dynamic executions and real-time behaviours which the static simulation is not able to do. Hence, dynamic simulation is widely adopted by software simulators with models of other specific modules to develop high-level abstractions for system performance estimations. Additionally, the system models also consist of two categories: functional models and performance models. Functional models emulate the functionality of different programs in a system while performance models interpret real-time behaviours of system components and give relative dynamic performance statistics. Usually, performance models are built up on the functional ones, called execution-based simulations, but the correctness of both kinds of models decides the results accuracy. All the native execution-based methodologies introduced above belong to the class of execution-based simulation.

Specifically in this thesis, accurate NoC behavioural models and performance estimates have been developed using advanced modern design automation techniques to generate fast, accurate and reliable system-level modelling at the early design stages. In the subsequent sections, specific factors of simulation performance will be introduced to expose their significance and impact on NoC system designs, which helps determine major relevant metrics for further research.

2.2.2 Major Performance Concerns in NoC Systems

To accurately develop high-level network behaviours and precisely simulate concerned system performance, it is important to figure out which factors may influence NoC architectures and are of particular interest to designers. How to present these factors using DA techniques in system-level evaluations to fulfil specific requirements is another question that needs to be considered. In this thesis, energy and timing issues and their trade-off balance are of a special concern for their substantial impact on system performance [20] [32] and thus discussed in next sections. Other factors like fault tolerance and scalability are also investigated in Section 2.2.2.3.

2.2.2.1 Energy

Energy consumption has become one of the most significant factors in modern NoC design due to the increasingly severe bottleneck of power supply systems [20]. It is manifest that the energy issue influences network design greatly and limits other performance since each part of system is an electronic component driven by electricity. Even where other auxiliary supplies are investigated, like that introduced in [86], for attempting to improve the power supply system itself, stability problems still hinder such the introduced method becoming a prior design choice. As per the contradiction between incremental power demands on the network communication and limited progress in power supply improvement, low-power design turns out to be of the highest concern in all aspects of distributed interconnection systems [27].

Energy consumed by the activities of interconnection networks has a cost in three main domains: *data generation*, *data communication* and *data processing* [20]. Of all three, the communication domain usually consumes the most energy because data transfer through node networking leads to most switching activities and the communication complexity linearly increases with the network complexity. Even in short-range communications, frequently sending and receiving data will cost substantial energy.

In major chip designs, various functional units are interconnected in the chip network and the power resource may be strictly limited. In this case, the completeness of the network topology strongly impacts on energy performance. In particular, it has a significant effect on network communication efficiency since the topological changes due to disconnected network parts may cause severe re-routing and re-organisation of the network. Besides, a precise detection on energy cost of network activities is of great help in emulating realistic performance of applications in a simulator [27].

In view of the importance of power savings in distributed network simulation, much research on low-power design is dedicated to either give possible solutions to mitigating traversal cost, like low-power routing protocols, or developing accurate energy evaluation models, like in [87]. In [88], current approaches to power management and conservation of distributed systems have been conducted in 3 main categories: *duty cycle*, *data-driven* and *mobility*. Duty cycle indicates the active time fraction of nodes during their lifetime. This approach puts nodes into sleep mode when there is no data incoming and wakes up nodes as soon as outside communication is needed. It ensures the most energy-expensive parts of network activities are activated for the shortest period to save the most energy. Data driven aims to reduce the total amount of data processed in order to lower the energy cost of the data-process part. The mobility approach reduces the total hops of data traversed from one node to its destination node by making partial relaying nodes movable.

These aforementioned approaches can greatly simplify the communication complexity of networks and thus lower the energy cost of traversed data in different ways. Nonetheless, choosing suitable power-saving design approaches and energy-detection models on the basis of specific design and application requirements should not be overlooked.

2.2.2.2 Timing

Timing is another important factor for distributed network designs. Since quicker response to applications is always the pursuit of network simulations, precise dynamic time statistics of network activities is very significant and necessary for custom designs. It can also greatly help establish network models with accurate descriptions of system performance [85]. Additionally, step-by-step timing records enable designers to diagnose simulation problems and trace detailed activities. Yet it is not easy to accomplish since too many detailed timing traces may cause redundant transmissions and consume extra energy.

Time synchronisation of different devices is highly demanded of many distributed computing systems [27]. It is not only an indispensable condition for normal running of networks but also a guarantee of the quality and precision of other metrics like the accuracy of duty cycling. Especially for ultra VLSI designs, partial modules in a network operating asynchronously to other modules may lead to unexpected simulation errors, module malfunction and system failure. However, as the complexity of components integrated into a single system continuously increases, global time synchronisation of the whole network becomes a bigger challenge to designers in terms of feasibility, applicability and energy cost issues.

In this case, asynchronous timing designs for different groups of devices in one network have gradually aroused attention from researchers in recent interconnection network designs [20] [14]. As tremendous numbers of modules integrated into a single chip may make global time synchronisation unfit or extremely difficult to realise, conditional or partial insertion of two or more sub clocks to drive different groups of modules under one global system clock has been investigated and considered to be a promising solution to fulfil timing requirements of distinctive applications.

This method can synchronise intra-group modules and eliminate potential clock jitter for those module groups because all the sub clocks are derived from single global clock. On the other side, it may also need new timing protocols to coordinate the inter-group sub-clocks with a global clock for eliminating potential clock skews. To our best knowledge, a general design approach to distinguish and adopt precise global time synchronisation and appropriate asynchronous sub-timing division would be highly desirable but still remains absent.

2.2.2.3 Other Factors

In many cases network processors may fail to work due to lack of power, environmental damage, component malfunction, software disability or sabotage [27]. The ability of a network to sustain data transfer and processing in the presence of failures is called **reliability** or **fault tolerance**.

The fault tolerant design of distributed systems requires three main aspects for consideration: *fault model*, *repair mechanism*, *fault detection and diagnosis* [20]. Furthermore, three levels of faults occurring in interconnection networks are *component-level*, *node-level* and *network-level* faults, which need to be considered from a whole viewpoint. According to [89], the fault models could be divided into four groups: stuck-at-value faults, calibration faults, additive faults and random noise faults. Current approaches of component-level fault detection and diagnosis could be geometric based, non-geometric based and *Bayesian's belief network* (BBN-based). Node-level detection and diagnosis approaches are divided into centralised-based and distributed-based detection. Connect-based and convergence-based fault repairs have already been developed by researchers [20].

The fault tolerance of networks can also be boosted by software methods particularly in failures due to hardware disconnection or damage [8] [83]. Research on self-adaptive routing algorithms like PROM [13] has been proposed to investigate such a possibility. This sort of algorithm could automatically re-route or re-organise data traversal paths in a network when a hardware failure or disconnection is detected. Yet such algorithms may compromise transmitting efficiency for reliability improvement, since the path selection mechanism is oblivious and at each node of a traversal path they randomly select the next hop direction. Although this mechanism may cost more time and energy than normal routing algorithms, it is still very suitable for specific applications where network stability is in high demand. In general, the optimised fault-tolerant solution to common interconnection networks is still undiscovered. The significance of reliability to a distributed system is beyond doubt and drives researchers for further investigation.

Despite the aforementioned design factors, there are still other important factors that may have a large influence on interconnection network designs and need to be considered. Based on the relevant factors listed in [4], [8] and [32], we have concluded other factors for NoC designs in the following:

Scalability: New schemes are needed to fully utilise such a large number of network processors and remain open for potential functional extensions [90].

Production cost: It is important to lower the cost of each module to keep the total cost of the network designs at an acceptable level.

Network topology: It is a challenging task to keep the network topology quasi-static and reduce its change frequency in NoC architectures in order to reduce the possibility of node failure and increase network service efficiency. Besides, different topologies adopted for various applications could influence system performance.

Security: In many cases, the network security, not only the information security but also the device security, is very important. Yet this service is usually at the cost of other services.

Quality of Service (QoS): This feature is at an early research stage since most research still focuses on low-cost and small-size designs. But properties such as network availability, transfer latency, throughput, and packet loss have shown high research values and promising application prospects.

From the above discussion, it is clear that low-power design is the most significant research position for designers. In NoC designs, the communication energy cost has increasingly important impact on global energy performance and then forces other modules to put low-power design as their first priority. However, other services like timing, security or reliability may be of the same significance as energy performance depending on specific applications and design requirements. Thus, the preferred plan for distributed network designs should evaluate all required characteristics, making trade-offs on them and offering optimal products according to specific applications. Although developing such models for system-level timing, power and other performance is very challenging, it is still necessary, since it will provide sufficiently accurate system estimations to explore the design space at a very early design stage, such that the global productivity is improved.

2.3 A Survey of NoC Application Mapping Techniques

Intelligent task allocation of various applications for specific NoC architectures is significant for achieving desirable system performance of target designs. As technology develops, more complicated tasks need to be dealt with in modern applications, requiring larger network size and more cores to process increasingly huge amounts of data. The more the network cores are collaboratively used for complicated tasks, the higher the computational complexity will be. Such an increase in computational complexity gives rise to longer search time because the total feasible possibilities of task allocation rises exponentially along with the core number, which may cause unacceptably long processing time for finding the optimal allocation. Consequently, the more processing cores a complex application requires, the less likely its optimal task mapping can be found within a tolerable time period. This long search makes the application mapping problem an instance of constrained quadratic assignment problem [91], which is known to be an NP-hard optimisation problem [91] [92] [93].

2.3.1 The Classification of Mapping Techniques

The application mapping techniques in NoC design cannot produce the generalised optimal task allocation that is suitable for any application since the various and specific design requirements may not be achievable concurrently or even contradict each other. Moreover, it is also hard to obtain the optimal task allocation to one specific application or application set if it is so complicated that the time cost for seeking the optimal solution is intolerable. However, finding the optimal allocation solution for specific applications is indeed essential to acquire advanced performance especially relating to energy- or timing-constrained issues. As a result, many different task mapping techniques have been developed in the past decade for optimal task allocations to specific applications in diverse NoC architectures. If the optimal task allocation is impossible to find within a tolerable processing time by the task mapping techniques, near-optimal or limited-optimal task allocations will be developed to asymptotically achieve as good as possible system performance.

For all the exact-optimal or near-optimal task allocations, the proposed mapping techniques can be classified as follows. Normally task mapping problems can be performed either on-line or off-line. As per [93], they can be classified as *dynamic mapping* and *static mapping* depending on the time at which the tasks are assigned to the IP functional blocks for data processing. Based on this division, a more detailed and specific classification of modern mapping algorithms under these two main categories is elaborated in [15] and listed in Figure 2.17.

As per the figure, *dynamic mapping* can be called on-line mapping [93]. By using it, the assignment and ordering of tasks are performed during the run time for executing

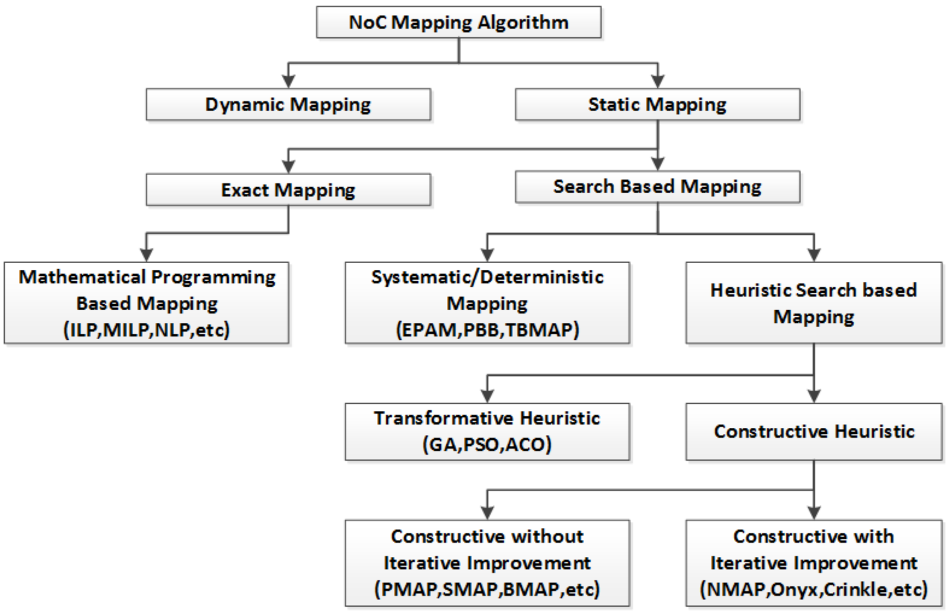

NoC Mapping Algorithm
Dynamic Mapping
Static Mapping
Exact Mapping
Search Based Mapping
Mathematical Programming Based Mapping (ILP,MILP,NLP,etc)
Systematic/Deterministic Mapping (EPAM,PBB,TBMAP)
Heuristic Search based Mapping
Transformative Heuristic (GA,PSO,ACO)
Constructive Heuristic
Constructive without Iterative Improvement (PMAP,SMAP,BMAP,etc)
Constructive with Iterative Improvement (NMAP,Onyx,Crinkle,etc)

Figure 2.17: The Classification of Mapping Algorithms [15]

applications. Dynamic mapping always tries to detect the emerging performance bottleneck of network traffic and distributes the real-time workloads to those lightly-load processing cores. Since this sort of application mapping depends on the current load of the processors, it ought to result in a better task distribution and network performance. However, the computational overheads this mapping technique causes may considerably increase the delay and energy cost of applications at run-time. On the other hand, the mapping of tasks in *static mapping* is generally decided off-line before the application starts running. For a given application and a target NoC architecture, static mapping always tries to define the best placement of tasks at design time. As the mapping is completed before execution, the mapping algorithm is executed only once. To map tasks onto NoCs, static mapping is highly recommended, since excess communication overheads in dynamic mapping will significantly affect the NoC system performance, which thus increases the overall system delay.

Since the application mapping and scheduling problems in NoC design are NP-hard, different mapping techniques have been developed depending on different practical scales of task-allocation problems. In static task mapping techniques, *exact mapping* has been developed using mathematical programming methods. Mathematical Programming minimises or maximises the objective functions by fulfilling multiple constraints of various problem variables. Currently popular Mathematical Programming methods include Integer Linear Programming (ILP), Non-Linear Programming (NLP) and Mixed Integer Linear Programming (MILP) [93]. The objective functions and constraints of ILP and MILP methods are linear and their full or partial variables are integers. An NLP method has non-linear objective functions or constraints. The MILP method is NP-complete while the mapping algorithms of ILP and NLP methods have polynomial complexity [94]. Another class of static mapping techniques is called *search based mapping*.

Two subclasses of search based mapping techniques are further given as *systematic/deterministic search mapping* and *heuristic search mapping* according to the distinctive search types and results. Smaller-size application mappings could use the deterministic search method to produce the optimal task allocation. Deterministic search mapping could exhaustively explore the whole search space of the solution set and then return the theoretically optimal allocation. One extensively used example of this search mapping is the Branch-and-Bound (BB) algorithm [92]. Heuristic search mappings are used in larger-size application tasks whose search time for the optimal mappings grows exponentially with the increase of task size, such that exhaustive search and deterministic search techniques are inefficient to be used. Heuristic search mappings support pseudo-random search in the exploration space of solution set, based on learned experience to produce the optimum. The generated optimum is only a reasonable task allocation obtained within a tolerably short time period.

Heuristic search mappings can be either application-specific or general-purpose to solve different-size task-mapping problems using *transformative heuristics* or *constructive heur-istics*. Transformative heuristics like Genetic Algorithms (GA) [95] transform some existing solutions to obtain better task allocation by attempting to explore an enlarged search space of solution set. Constructive heuristics produce partial solutions sequentially until the final complete solution is reached. If there is no position change of cores selected based on certain pre-defined standards, it is a constructive heuristic without iterative improvement. If the iterative improvement of core positions is performed upon an initial core allocation based on pre-defined standards, constructive heuristics with iterative improvement are obtained. The detailed previous work of these proposed mapping techniques will be discussed below.

2.3.2 Dynamic Mapping Techniques

As introduced, **dynamic mapping** is an on-line mapping technique that executes the application tasks during run time. The task placement on NoC cores is altered during the task execution to ease the performance bottleneck caused by heavy traffic loads while fulfilling other specifications such as congestion-aware, communication-aware and energy-aware requirements.

Chou *et al.*[96] have proposed an efficient technique for run-time application mapping onto an NoC with multiple voltage levels. It minimises the communication energy cost as much as 50% compared to arbitrary mapping solutions while still fulfilling the required performance. Moreover, this heuristic can be easily scaled for large-size applications.

Based on it, Chou and his colleagues have proposed an energy- and performance-aware heuristic in [97] for incremental application mapping onto NoC Networks with multiple voltage levels. It targets real-time applications that dynamically map on NoC platforms and connected resources with multiple voltage levels.

Additionally, Chou also proposed an user-aware dynamic task allocation in [98]. It is a run-time task mapping strategy in homogeneous NoC platforms with the user behaviour information incorporated in the resource allocation. This strategy provides good dynamic response and adaptivity of task mapping upon user needs. Experimental results have shown about 60% communication energy savings of this dynamic task mapping compared to random task mapping.

A heuristic Dynamic Spiral Mapping (DSM) algorithm for 2-D mesh NoC topologies has been proposed in [99]. It comprises of two different approaches: Full Dynamic Spiral Mapping (FDSM) and Partial Dynamic Spiral Mapping (PDSM). The heuristic places application tasks on to different network cores following a *spiral* path from central cores to boundary cores of the mesh topology. The task mapping at design time is executed by the Static Spiral Mapping (SSM) [100] during run time. Experimental results have shown the PDSM has less reconfiguration time than FDSM in 82% of the simulation cases.

A set of heuristics for dynamic application task mapping has been presented in [101] and [102]. The work targets minimising the network congestion by investigating the performance of task mapping heuristics on NoC platforms with dynamic workloads. The heuristics firstly set an initial task mapping and then dynamically map new tasks upon communication requests by using one of the following techniques: First Free (FF), Nearest Neighbour (NN), Minimum Maximum Channel load (MMC), Minimum Average Channel load (MAC) and Path Load (PL). Experiments show congestion reduction are achieved when congestion-aware mapping heuristics are employed. The PL mapping results in better optimal solutions than MMC and MAC heuristics.

In [103] and [104], a group of communication-aware real-time mapping heuristics for the efficient mapping of multiple applications onto an heterogeneous NoC platform have been proposed. The heuristics support more than one task mapped onto one single processing element (PE) by checking the available resources in advance. If the unmapped tasks exceed available PE resources, adjacent communicating tasks are recommended to be mapped onto the same PE. The proposed heuristics can be deemed the extended work of [101] and [102], which employs a packing dynamic mapping strategy for minimising the communication overhead and algorithm execution time in the same NoC platform. Experimental results show the heuristics can obtain a reduction in overall algorithm execution time, energy consumption, average link load and latency compared to other contemporary dynamic mapping heuristics.

An energy-aware heuristic for dynamic application task mapping has been proposed in [105], named Lower Energy Consumption based on Dependencies-Neighbourhood (LEC-DN). The LEC-DN heuristic executes the task mapping in a real RTL-modelled NoC platform, giving rise to accurate evaluation results. Real applications are evaluated as benchmarks, showing the proposed LEC-DN heuristic may reduce the communication energy up to 22.8% compared to other dynamic mapping heuristics.

Mandelli *et al.*[106] have extended the work, allowing multiple tasks assigned to one single processing element instead of single-task mapping. Experiments present 51% energy savings on average and 18% algorithm execution time overhead on average for the multi-task mapping over single-task mapping.

2.3.3 Static Mapping Techniques

Static mapping techniques allocate network resources to various application tasks before the task execution starts. The determined task mapping scheme is not going to be changed or modified thereafter. Compared to dynamic mapping, static mapping is an off-line mapping such that all the allocations of network cores and routers to different application tasks are given at design time. Various techniques have been developed to look for the optimal or near-optimal/suboptimal solutions while fulfilling other performance constraints or design specifications.

2.3.3.1 Exact Mapping

Exact mapping uses mathematical programming methods to produce the optimal mapping of application tasks. Mathematical programming methods minimise or maximise the objective functions by fulfilling various constraints of distinct and specific problems. Previously proposed mathematical programming methods for optimal task mapping mainly use Mixed Integer Linear Programming (MILP) techniques.

In [107], a MILP model has been proposed that decides a task mapping optimising the trade offs between algorithm execution time, processor resource and communication cost for heterogeneous multiprocessor systems. It may be the the first to propose a mathematical programming-based task mapping for multiprocessor systems. It is a hardware/software co-design and total communication delays are met. A media application of H.261 processing is used as the practical experiments to successfully show a correct task and schedule allocation by the proposed MILP model and constraints.

A many-to-many Core-Switching Mapping (mCSM) has been proposed in [108] to investigate the Core-Switching Mapping (CSM) problem. It optimally maps cores of an NoC architecture for some real and random application tasks to minimise both the energy consumption and the network congestion during task execution. The proposed mapping, as stated, was the first to offer an MILP formulation for the complex CSM problem, encompassing the suboptimal solutions of core placement, switches for cores and communication traffic. The experiments compared the proposed mapping with previous one-to-one mappings, indicating a 81.2% energy saving and 2.5% bandwidth saving.

The authors of [23] have presented an integrated design methodology to automate all design steps of application-specific NoCs including core mapping onto NoC topologies, physical planning (computing core position and size), topology selection, topology optimisation and instantiation. They have also presented a design methodology to guarantee Quality-of-Service (QoS) for the application by fulfilling delay and real-time constraints of the network traffic in the physical planning process. A greedy mapping is initially used to obtain core mapping onto a specific topology, then compute the traffic routing between cores to improve the core and switch positions by iteratively using a robust tabu search. An MILP-based algorithm has been developed to refine the design area and power as well as satisfying the application QoS. Experimental results have shown up to 2 times area savings, up to 5 times bandwidth savings and from 1.6 times (switches) to 3.8 times (number of wires) network resource savings of the proposed method compared to traditional approaches.

A novel MILP formulation for synthesis of custom NoC architectures has been presented in [109]. Its optimisation objective is minimising the power consumption of custom NoCs subject to specific performance constraints. The custom NoC synthesis problem has been addressed by using a two-stage MILP formulation that is split into two subproblems: system-level floorplanning with the objective of power cost minimisation subject to the layout constraints, and custom NoC topology generation with the objective of power cost minimisation subject to the performance constraints. Since the optimal MILP formulations for the two stages have timed out for quite a few benchmarks, there has also been a clustering-based heuristic technique designed in this stage to reduce the unacceptable execution speed caused by the MILP formulation. On average, experiments have produced 15% energy savings and 4% resource savings of clustering-based heuristics compared to original optimal MILP formulations.

Since energy issues are becoming critical in modern chip design industry, much work focusing on energy-aware mappings has been presented. Representative exact mappings for **energy** are introduced below. The work in [110] presents an ILP (Integer Linear Programming) based formulation of the problem for minimising the communication energy in NoC based CMPs (Chip Multi-Processors) since it is increasingly important to manage high-level communication and power control. The presented formulations can select the optimum link set as well as the voltage and frequency of these links, such that the problem objective of minimising energy consumption can be achieved. Experimental results have shown that the ILP based solutions produce impressive energy reductions.

An unified approach utilising a Mixed Integer Linear Program (MILP) formulation for the optimal solution and MILP relaxation as well as randomised rounding for the heuristic to map application tasks onto heterogeneous NoCs with multiple voltage operation has been presented in [111]. The proposed mapping aims to minimise energy costs subject to performance constraints. It involves a unified solution of several subproblems without compromising the solution time and proposed techniques for optimal and heuristic solutions. Experiments based on E3S benchmarks and real media applications have resulted in the near-optimal solutions produced by the proposed heuristic in a fraction of time required by the optimal solution.

Based on the work in [111], Huang *et al.*[112] have extended the former existing Integer Linear Programming (ILP) formulation to consider both processing and communication energy consumption, exploring the trade off between these two major energy consumers to seek the optimal mapping from a system point of view. Thereafter, a Simulated Annealing with Timing Adjustment (SA-TA) heuristic has been proposed to accelerate the extended ILP optimisation process. The work has reached the goal of developing efficient algorithms for computing system-wide energy-aware task mapping onto heterogeneous MPSoC systems. Experiments have proved the extended ILP formulation has considered the trade off between processing and communication energy cost but has a slow execution speed to yield the system-level optimal mapping. The SA-TA heuristic has then been validated to give a considerable improvement in the execution speed while producing near-optimal solutions, very close to the global optimum.

A novel $0-1$ Integer Linear Programming (ILP) formulation for application task mapping onto Mesh based NoCs has been proposed in [113], aiming to obtain optimal results for the system with the minimisation of energy consumption in a tolerable execution speed. The proposed method has been tested on six different multimedia application benchmarks to achieve optimal or near-optimal results within the given time limit. It is manifest that, however, the **network link bandwidth constraint** is not considered in the work. The reported CPU execution speed for different application benchmarks is too slow to obtain the optimum by using the proposed method.

To overcome the high algorithm execution time problem, the same author has proposed another work with the introduction of a clustering-based ILP relaxation [114]. The proposed method uses the previous $0-1$ ILP formulation to establish optimal task mapping. If the size of application tasks is too large to achieve the optimum with tolerable execution speed, the proposed method represents the application by partitioning original task graph and mesh network into smaller sub-graphs and sub-meshes to decompose the solution space into smaller polyhedrals. The partition schemes are either mesh cut partitioning or graph clustering as in [109].

Traffic contention in networks during task execution has become more common as the amount of processing data has increased dramatically in current industrial designs. The work in [115] is one typical example of exact mapping concentrating on this issue. In this work, the impact of network traffic contention on the application task mapping for tile-based NoC architectures has been analysed using an ILP formulation of the contention-aware task mapping problem. This mapping technique has obtained the optimal solution to application tasks with minimisation of the inter-tile network traffic contention. The ILP approach followed by a mapping heuristic have been proposed together to produce the near-optimum while satisfying the runtime of application execution. Experiments have shown the proposed mapping technique gives a considerable decrease in packet latency with a negligible communication energy overhead.

As observed from the above discussion, the common computational bottleneck of exact mapping (in particular, the Linear-Programming (LP) based mathematical methods) is the **scalability problem** such that the overall algorithm execution time of generating the optimum of large-size application tasks is intolerable. To alleviate it, many techniques have been used at the cost of result accuracy reduction (near-optimal or suboptimal solutions) and other resource overheads such as energy, area, network resources, communication latency and so on.

2.3.3.2 Search Based Mapping

Another type of static mapping technique is *search based mapping*, in which various searching techniques are adopted in the possible solution set to find the optimum. For the *systematic/deterministic search based mappings*, the specific searching technique is that the whole search space in the possible solution set is exhaustively explored to reach the theoretical optimum. *Heuristic search based mappings* utilise pseudo-random searching techniques in exploring the potential solution set based on learned experience but may return reasonably optimal, near-optimal or suboptimal solutions due to the limited algorithm execution time for large-size applications.

Systematic/Deterministic Search based Mappings:
Branch and bound algorithms are usually used in *systematic/deterministic search based mapping techniques.* They systematically search each option of all potential topological mappings in tree branches while bounding the unallowable choices to seek the theoretical optimum. In [92], [22] and [116], the authors have proposed a set of energy- and performance-aware task mapping techniques using the branch and bound algorithm for tile-based regular NoC architectures under specified performance constraints through bandwidth reservation.

In [92], an energy-aware topological mapping algorithm has been developed for generic regular NoC architectures. It minimises the total communication energy cost while satisfying specific system performance through bandwidth reservation. The algorithm firstly formulates the task mapping problem in a topological way, then solves it using an efficiently modified Performance-aware Branch and Bound (PBB) algorithm. Several speed-up techniques have also been designed for accelerating the searching process. Some real-world media experiments have shown the proposed Energy- and Performance-Aware Mapping (EPAM or GMAP) algorithm combined with PBB algorithm resulting in an average 60.4% energy savings compared to an ad-hoc implementation.

Besides the proposed EPAM algorithm, the authors have also expanded the search space and improved the solution quality by exploiting the routing flexibility in regular NoC architectures in [22]. After the EPAM problem is formulated, a novel routing path allocation scheme has been produced for the upcoming solving by PBB algorithm with the objective of efficiently finding out deadlock-free, minimal routing paths while balancing the network traffic [22]. The scheme has constructed a Legal Turn Set (LTS) including west-first and odd-even adaptive routings, and XY deterministic routing to allocate flexible routing paths within reasonable computational time period. The EPAM algorithm and flexible routing path allocation scheme have been summarised together in [116]. Moreover, two extensions based on previous work have been proposed to enable the EPAM or GMAP techniques associated with the PBB algorithm applicable to the specific NoC architectures with irregular-size regions or pre-mapping IPs.

The above mappings are designed for NoC architectures with a single IP core connected to a single router, which may cause heavy traffic loads in networks when IPs have large communication volumes. This may further result in chip **reliability problems** due to the high power density of those traffic hotspots. [117] has proposed a novel mapping to solve this problem. In the work, new Network Interfaces (NIs) have been proposed, initially comprising several new styles of NIs. Furthermore, a Traffic-Balanced Mapping algorithm (TBMAP) has also been proposed based on the new NIs. TBMAP uses a modified branch-and-bound searching algorithm as given in [92], [22] and [116], mapping tasks with various types of NIs onto 2D-mesh NoCs. The TBMAP has a short algorithm execution time to obtain more balanced, decentralised traffic loads without sacrificing the network performance. Instead, non-minimal data routing paths in some cases become the trade-off for more balanced traffic loads.

The authors in [118] have presented a fast power- and performance-aware task mapping technique, Elixir, targeted at computing the lowest communication cost with minimum average latency and power cost. The mapping combines both the bandwidth-constrained mapping [119] (NMAP, which will be introduced in later section *Constructive Heuristics with Iterative Improvement*) and branch and bound search algorithm as stated in [92], [22] and [116]. Elixir mapping initialises the NMAP mapping to produce solutions, then proposes a 2-step algorithm to refine the results. Experiments have shown the Elixir mapping algorithm generates better mapping solutions of real-world media applications than EPAM and NMAP in terms of average latency, power cost and communication cost.

From the aforementioned examples, the branch and bound search based techniques could reduce the overall algorithm execution time of task mappings significantly while producing the theoretical mapping optimum, compared to exact mappings. Yet these algorithms may need large buffers and long CPU execution time for searching for the optimal solution of large-size application mappings.

Heuristic Search Based Mappings:
Many heuristic methods have been presented to solve complex application mapping problems. They can be roughly divided in *Transformative Heuristics* and *Constructive Heuristics*.

Transformative Heuristics:
Transformative Heuristics usually transform and refine certain existing mapping algorithms in order to acquire better mapping solutions. Several typical evolutionary techniques such as *Genetic algorithm* (GA), *Particle Swarm Optimisation* (PSO) and *Ant Colony Optimisation* (ACO) have been reported.

Genetic Algorithm (GA) based Heuristics
A genetic algorithm is a stochastic search algorithm derived from natural processes. By using natural selection, a certain population of chromosomes has evolved for several generations, such that the offsprings contain new features caused by either crossover from two parent chromosomes or mutation from random portion changes from the parent chromosomes. For evolving the desired optimal results, both the crossover and mutation operate at controlled rates. The termination criterion of evolution using GA algorithm is also set upon specific baselines, such as after a certain number of generations or no improving evolution in the latest several generations.

A 2-step GA based mapping technique for Mesh-based NoCs has been reported in [95]. It builds a specific communication delay model to minimise the system delay and overall algorithm execution time especially for large task graphs. By using the proposed techniques, near optimal solutions to heterogeneous application tasks can be derived in a short execution time. Specifically, system delay models are initially setup, followed by a two-step genetic algorithm to solve the vertex mapping problem. Two genetic algorithms are designed, one for each step.

In [120], a delay computing-model has been presented with a genetic algorithm for application mapping in 2D-mesh NoC architectures. The GA-based algorithm achieves near optimal solutions to core mapping with a minimum average delay. In this work, the delay-computing model captures different data routing probabilities and traffic contention. Then a genetic algorithm is utilised, based on the proposed average delay model to obtain optimal solutions within a reasonable time. Experiments with random traffic in various-sized NoCs have shown an average 20% execution time reduction for the proposed algorithm compared to random mappings.

A GA-based multi-objective approach to optimise application mappings in Mesh-based NoC architectures has been proposed in [121]. It is an efficient and accurate approach based on evolutionary computing techniques, specifically genetic algorithms (GA-based), to achieve an approximation of the Pareto mapping set that maximises the performance and minimises the power cost. As the authors state, the work is the first to address the task mapping problem by using a Pareto-based multi-objective optimal approximation. With the integration of an exploration framework, including the kernel of an event-driven, trace-based simulator, the proposed technique requires a very limited number of configurations to provide an accurate Pareto approximation from the optimal solution set.

Additionally in [122], the same authors have compared the work with two widely-used, extended mapping approaches (Branch and Bound and NMAP) in terms of the accuracy and efficiency of their results, in order to explore the mapping space under a multi-criteria consideration. In particular, they have extended the Pareto-based branch and bound approach and the Pareto-based NMAP approach for multi-objective optimisation and the generation of Pareto-optimal solutions. Another extension of the previous work has emerged for taking account of dynamic effects in task mapping and evaluation of their simulating framework.

MOGA, a Multi-Objective Genetic Algorithms based technique has been presented in[123] to find the optimum from the Pareto-optimal solution set for application mapping on regular tile-based NoC architectures. MOGA is an energy- and bandwidth-aware topological mapping technique aiming to minimise the energy cost and link bandwidth requirements. One-one as well as many-many switch and task mappings have been used in the proposed technique. Random benchmarks and real-world applications have been experimented with to evaluate MOGA techniques and demonstrated 70% energy savings and 20% link bandwidth savings for the proposed work.

CGMAP, a GA based application mapping technique has been presented in [124] and [125] for Mesh-based NoC architectures. This technique has used a novel Chaos-Genetic algorithm for mapping IP cores with the objective of Quality of Service (QoS) improvement in NoCs. Chaotic behaviours are none-periodic, long-term and sensitively dependent on initial conditions in deterministic systems. They also have important dynamic characteristics like pseudo-randomness and ergodicity [124], where the latter characteristic guarantees a chaotic variable to traverse ergodically over the whole exploration space. This intrinsic feature of a chaotic process makes it promising to replace the random-based optimisation process in genetic algorithms. Experiments with both synthetic traffic and real-life applications have shown the proposed CGMAP technique performs as well as other existing mapping techniques in terms of hop distance, latency and energy [125].

The GA-based mapping algorithms may suffer from **slow convergence** as they evolve a large group of generations to reach the optimal solution. Besides, since the final offspring in the evolving solution set is often deemed the optimal one, it is quite easy for genetic algorithms to achieve a **locally optimal solution** instead of a global optimum, which impairs the accuracy of the techniques.

PSO based Heuristics

Another transformative heuristic search based mapping is the Particle Swarm Optimisation (PSO) heuristic [126], which is a population-based stochastic search technique inspired by social behaviours of bird flocks. In a PSO heuristic, each candidate solution is a particle containing a fitness value in the search space. Many particles coexist and collaboratively evolve in search space based on the experience of tracking and memorising the best encountered positions from their own and their neighbours until optimum is reached. The fitness value of particles determines the solution quality.

PLBMR is a PSO-based mapping and routing technique for Mesh-based NoCs [127]. It aims to minimise the communication energy and normalised worst link load. Two phases are involved in the proposed PLBMR mapping. PSO has been used in both phases by building the particle structure and initialising particle generations where the particle configurations are derived from GA-based models introduced in [120]. Experiments have compared the proposed PLBMR with random mappings, branch and bound mapping and GA-based mapping, indicating a variable technique of PLBMR in mapping problem solved with balanced traffic load and minimum communication energy.

ACO based Heuristics

The Ant Colony Optimisation (ACO) based heuristic mapping is also a class of transformative heuristic search based mappings. The ACO technique [128] is a population based probabilistic technique inspired by the cooperative behaviour of ants to establish highly-structured routing paths from their colony to a food source based on low-level information interactions. Once an ant has found a path to a food source from their colony, other ants, though having very limited interactions with others, are highly likely to follow the same path. This process is characterised by a positive feedback loop such that the probability of the same path chosen by other ants after one ant has chosen it increases with the number of other ants, which is eventually managed by the ants to establish shortest (optimal) paths.

An ACO-based algorithm has been used in [129] to map application tasks onto 2D-mesh based NoCs, such that the bandwidth requirement is minimised. This algorithm is a quadratic assignment problem to figure out near-optimal solutions to the formulated mapping problem. Certain numbers of ants, generations (iterations) and the probability between tasks and cores have been initialised to randomly map tasks to cores based on probability and tabu search. When all ants have been distributed, the one with the most optimised mapping solution will be updated until a near optimal result is achieved after predefined iterations. Simulation results have shown the proposed method reduces the bandwidth usage by 48% compared to random mappings.

Lately, some evolutionary techniques of Hybrid GA, PSO or ACO heuristic mappings have been researched on complex NoC applications. An Energy- and Buffer-Aware Mapping (EBAM) [130] gives a GA-based algorithm with a self similar traffic to jointly minimise energy and buffer on Mesh. A hybrid mapping algorithm [131] based on PSO-GA and PAO-SA (Simulated Annealing) is proposed on 2D-mesh NoCs for pareto optimisation of performance and reliability.

Constructive Heuristics:
Constructive heuristics are another kind of heuristic search based mapping that generate partial solutions to an application task mapping problem consecutively. They try to constructively yield near-optimal solutions through carefully considering most characteristics of a mapping problem. Two types of constructive heuristics have been used for solving application mapping problems.

Constructive Heuristics without Iterative Improvement
This type of constructive heuristics produces an application mapping sequentially based on pre-defined configurations. It maps the core graph onto NoC topologies. Once the mapping is placed, there is no change to the core positions. In other words, there will be no iterative refinement included in this type of mapping heuristic.

Physical Mapping algorithm (PMAP) [132] is a two-phase application mapping algorithm for NoC based architectures. The algorithm divides a task graph into clusters and places them onto processors sequentially without backtracking to lower computational complexity. Experiments have tested the PMAP algorithm in producing mappings with efficient communication delays.

SMAP [133] is a simulation environment to address the optimal application mapping and task routing for 2D-mesh NoCs. The objectives of this mapping technique focus on minimising average hop distance and link bandwidth of the NoC architectures. In this algorithm, the core placement of the remaining tasks follows a spiral fashion from the centre to the boundaries of the mesh platform. Experiments have compared the SMAP algorithm with a GA-based algorithm and random mappings in execution speed, energy cost and algorithmic complexity.

BMAP [134], a binomial IP mapping and optimisation algorithm has been proposed for efficient hardware design of NoC infrastructures. The proposed BMAP mapping aims to minimise network traffic, hop distance and hardware cost. In the algorithm, a traffic model used for task mapping has been extracted using a greedy algorithm. Then three main operations are composed: binomial merging iterations, traffic surface creation and hardware cost optimisation. Experimental results have shown a 37% reduction of network traffic load, a 46% reduction of average hop distance and 51% to 85% reduction of hardware cost in BMAP compared to NMAP [119].

Chain-Mapping (CHMAP) has been presented in [135] as an efficient mapping algorithm for Mesh-based NoCs. It produces chains of connected cores to help prioritise IP core mappings. CHMAP considers the communication volume of a serial core instead of a single IP and generates specifically prioritised core mappings. The algorithm has been divided into four phases. Experiments with real-world media applications have found the performance of bandwidth cost by CHMAP is acceptable in comparison with other mapping algorithms.

Constructive MAP (CMAP) is a fast constructive heuristic algorithm that has been proposed in [136] for core mapping onto NoC architectures. It aims to minimise the total communication cost and energy consumption of application tasks. This algorithm can be used in any size of NoCs as well as many topologies other than the given mesh network by modifying its evaluation function. CMAP is a hybrid algorithm combining two mapping algorithms: Link Based Mapping (LBMAP) and Sort Based Mapping (SBMAP). The solutions from these two mapping algorithms are always compared to select the best one as the near optimal solution. The accuracy, efficiency and scalability of proposed CMAP algorithm has been validated using real media applications.

Constructive Heuristics with Iterative Improvement
In this type of constructive heuristics, an initial mapping solution is constructed from the core graph each at a time based on pre-defined configurations. Then an iterative refinement is implemented upon the initial solution.

NMAP [119] is a fast application mapping algorithm for Mesh/Torus based NoC architectures, minimising the average communication delay under bandwidth constraints. It has been presented for mapping with single minimum-path routing and split traffic routing. Three phases are involved to obtain the near optimal solution by heuristics. Video processing benchmarks have been tried on a self-built simulation framework to validate significant savings of NMAP in bandwidth usage and communication cost compared to other existing algorithms.

Onyx, presented in [137], is a new bandwidth-constrained heuristic method for mapping cores onto Mesh-based NoC platforms. It minimises the hop counts between IP cores with less complexity, leading to an improved energy consumption. In Onyx mapping, unmapped cores are ranked upon their communication bandwidth. Cores with higher communication bandwidth have been mapped earlier, i.e. they have higher priorities for mapping. Real-life applications have validated the Onyx algorithm outperform other existing algorithms in communication cost.

Crinkle [138] is a heuristic task mapping algorithm that also aims to minimise the hop numbers between IP cores. The name 'crinkle' is derived from the crinkle moving pattern this algorithm uses for lowering the algorithm complexity. Priority lists including three task priority lists and one platform priority list have also been proposed to optimise the core mappings. Both synthetic and real applications have reported the Crinkle algorithm performs competitively in execution speed, energy cost and hop counts compared to other existing algorithms.

A two-step novel and efficient mapping algorithm, called Citrine, has been introduced in [139]. It produces optimal mappings with the objective of minimising communication cost and improving the fault-tolerance. A criterion called the vulnerability index has been defined to evaluate the routing fault tolerance in this algorithm. Citrine combines the work of Onyx routing path selection in [137] and Branch-and-Bound searching in [116] to form an efficient and accurate mapping algorithm. A total cost function for communication cost minimisation and fault-tolerance improvement of Citrine algorithm has been used to validate these features.

As observed, heuristic search based mapping techniques can always achieve acceptable, near-optimal or suboptimal solutions to various application task mappings in a reasonably short algorithm execution time while fulfilling other design specifications. Compared with other types of mapping techniques, the heuristic search based mappings have tackled the scalability problem of large-size application tasks at the cost of tolerable result accuracy.

2.4 A Survey of Current Network Simulators

A common problem of research on network simulations is that the types of interconnection networks are various and application-specific, such that there is not a generalised software platform to construct all those systems [85]. For instance, several generations of simulation platforms for Wireless Sensor Networks or Mobile Networks like SmartDust [140], Mica [141] and Telos [142] have been developed, but all of them are designed for modelling and simulation of specific types of networks, which causes high production costs and a very limited experimental range. NoC simulators have similar problems. To solve this, a brief overview of modern network simulators is initially presented as the prerequisite knowledge to subsequent research. After the review, a candidate simulator is chosen as the backbone platform to extend its functions for further research in the thesis.

2.4.1 Emergence and Current Classification of Network Simulators

As the technology has advanced, Electronic Design Automation (EDA) has developed rapidly [27]. Software and hardware co-design in most fields of interconnection network systems has become feasible and desirable since the fast rise of integration complexity in interconnected network devices has increased the cost of hardware. This requires improved automation techniques [14]. Whether current EDA techniques can emulate network activities and performance precisely enough for industrial product designs has received high research interest. Using software based design procedures to replace hardware design methods means that the total production cost and time to market can both be greatly lowered.

Research attempts have started to investigate this potential technical revolution. As a result, several simulators at different levels of architectural modelling prior to physical implementation for modern NoCs have been proposed [143]. With the proposed concept of NoC architectures, researchers have developed network simulators with on-chip interconnection models for evaluating NoC system performance. Those network simulators may originate from various sources. Typically, three main categories of simulators for NoC designs can be differentiated as per [144]: *regular network simulators*, *dedicated NoC simulators* and *full-system simulators*. Some models and tools are also proposed recently for specific NoC systems for emulating certain metrics or types of NoC applications. The classification of network simulators for NoC system designs is illustrated in Figure 2.18.

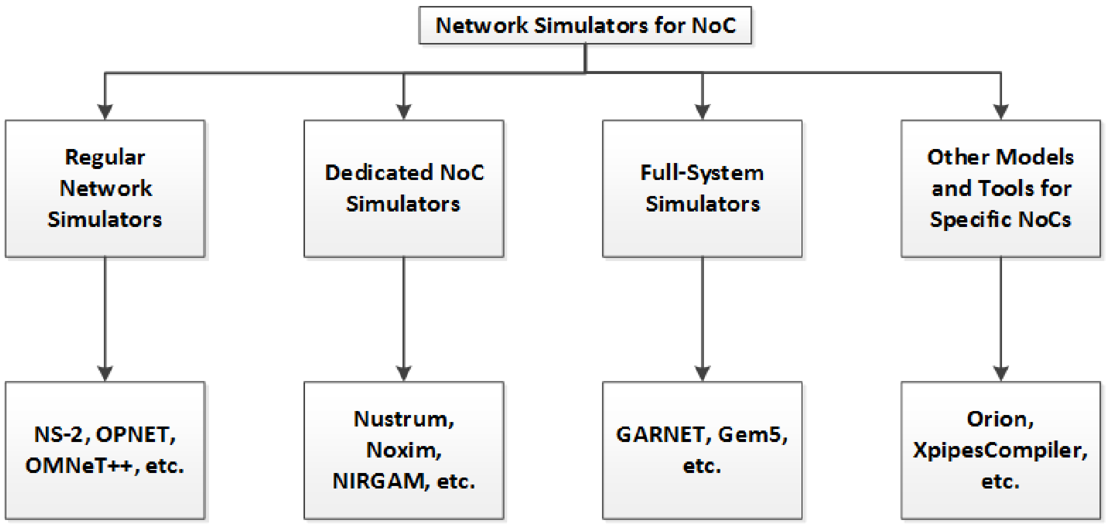

Network Simulators for NoC
Regular Network Simulators
Dedicated NoC Simulators
Full-System Simulators
Other Models and Tools for Specific NoCs
NS-2, OPNET, OMNeT++, etc.
Nustrum, Noxim, NIRGAM, etc.
GARNET, Gem5, etc.
Orion, XpipesCompiler, etc.

Figure 2.18: The Classification of Current Network Simulators for NoC Simulations

Regular network simulators are not normally designed specifically for NoC systems. They borrowed concepts from computer and communication networks, such as wireless sensor networks, mobile networks and server clusters, to simulate NoC systems by making use of the similarities between general networks and on-chip interconnection networks. **Dedicated NoC simulators** are typically designed for NoC simulations. They mostly abstract high-level models of NoC architectures to emulate data routings and processing at message level. Certain metrics, like latency, throughput, power and area of NoC applications, are popularly evaluated their performance by this kind of simulators with some chosen traffic patterns, topologies and routing algorithms. **Full-system simulators** model not only the network characteristics but also details of other layers like processing cores, cache hierarchy, cache coherence and memory systems. Such simulators enable the evaluations of exact technical applications on the entire NoC system, offering system-level optimisation and accurate simulation results.

2.4.2 Regular Network Simulators

Besides previously introduced simulators, some more simulators developed for WSN or Mobile Networks like TOSSIM [145] and WSNsim [146] are also feasible for NoC simulations. It is because the NoC architecture is similar to WSN [20]. The codes in these simulators for specific network formations, various routing algorithms and even the fault-tolerance models, such as the stuck-at-value model, could be well-suited for NoCs with a suitable modification to communication module modelling and routing mechanisms. Moreover, the development language for WSN simulations is the dominant high-level language in embedded system programming [83], which is also fit to high-level model extractions of NoC systems.

Network Simulator-2, or **NS-**2, has been widely used for network simulations [147]. It provides a large variety of models for network modules, topologies, communication protocols and traffic patterns. It also supports large numbers of network simulations with respect to simple power models. But the trace file for dynamic behaviour statisticsin NS-2 is overloaded and over-detailed. Moreover, its energy models are too simple to achieve accurate result, especially the dynamic power loss in ultra VLSI network designs. NS-2 uses the C++ language to develop its detailed protocol implementations with high switch speed [148]. The original version was designed for on-chip networks and a project called Sensorsim [149] was conducted to extend the NS-2 framework to support sensor network simulations.

OPNET [150] can generate an optimised hierarchical methodology automatically for application-specific NoC systems. It provides statistics of on-chip communication for cycle-accurate analysis and power estimations based on a standard library. It also leverages the existing tool to adapt to industrial-grade network modelling. The **OMNeT++** [151] simulator can be used to on-chip networks. It supports irregular topology construction and fast performance results in terms of average throughput and latency.

Regular network simulators may support flexible and custom NoC designs due to a huge variety of available protocols and topologies. But their simulation results are often less-accurate since many of them are not developed specially for on-chip interconnection networks. Some NoC characteristic models may be absent in these tools.

2.4.3 Dedicated NoC Simulators

Due to the limits of existing regular network simulators in NoC designs, researchers have been motivated to propose several dedicated NoC simulators. The NoC simulation tool proposed in [143] presents a SystemC-based NoC simulation environment. It supports dynamic trace information statistics for accurate behaviour modelling, which enables irregular network generation. Yet it has no energy models and only provides the wormhole routing method.

Another NoC simulator, **Nostrum** [39], gives a concrete packet-switched communication protocol stack for offering designers the possibility to customise the functionality of different structural layers from physical to transport layer with respect to specific applications. But the simulator lacks a precise energy model too.

A generic, modular and extensible simulator: Network-on-chip Interconnect Routing and Application Modelling (**NIRGAM**) is proposed in [25]. It is a discrete-event and cycle-accurate simulator specifically designed for NoCs. It provides substantial support for application-specific experiments on modular NoC platforms in terms of latency, energy and throughput performance. Various options at almost every stage of the NoC design process are available to be evaluated, which enables the simulator to be easily extended to include user-specific applications, routing algorithms, switching techniques and network topologies. On the other side, the NIRGAM simulator needs to improve its functionality on network simulations of irregular topologies.

Noxim [152] is another NoC simulator developed in SystemC/C++. It provides a number of user-customised parameters for application-specific simulations. The network metrics in terms of throughput, delay and power can be evaluated by the simulator. A special tool called Noxim Explorer is integrated for the design space exploration.

A dedicated NoC simulator written in Java [153] is proposed to offer message processing at flit level in mesh interconnection networks. It provides simulations with several popular routing policies, flow control methods and data collection ways. This simulator, as authors claimed, has the advantage of portability brought by Java in its NoC simulations.

NoCSEP [154], a framework called NoC centric System Exploration Platform uses a quasi-formal description language PACMDL to drive its application traffic generator, which configures the task-graph characteristics of any parallel application. This feature makes the simulator suitable to optimise application-specific NoC designs.

Dedicated NoC simulators can provide better simulation results of NoC designs than regular simulators due to their more sophisticated network modelling. But the system architectures other than network part may be less accurate since only high-level network models are parameterised, which negatively affects its practicability for full-system implementation.

2.4.4 Full-system Simulators

To accurately evaluate the NoC system performance, it is important to analyse the impact of NoC optimisation techniques on full system behaviours because not only network characteristics but also other micro architectures like memory inside the network impact the system performance. This fact urges the demand of detailed and accurate interconnection network models within a full-system framework.

GARNET [155] is such a full-system simulation framework that has a group of micro architectural modelling like five-stage pipelined routers with virtual channel allocation, a detailed timing model, a shared and private L2 memory system and other common network components. GARNET enables system-level optimisation as well as component modelling of other levels in detail, evaluating the application performance on an entire system instead of just the network. It can obtain correct simulation results of popular performance metrics such as timing and power.

Gem5 [156] is a flexible, modular full-system platform that can evaluate a wide range of computer systems. This infrastructure offers diverse models of CPUs, ISAs (Instruction Set Architectures) and memory systems with multiple execution modes. Currently Gem5 can support most commercial ISAs and capture detailed aspects of processing cores, cache hierarchy, cache coherence, and memory systems, making it widely used in both industry and academia. Its appealing features include high flexibility, wide availability and great utility.

Full-system simulators can give the most accurate simulation results of NoC systems amongst all kinds of simulators. Yet the operating efficiency of such simulators are normally the lowest due to their high-complex and thorough system models that are difficult to accurately implement with fast execution of realistic workloads. Therefore, the balance between model complexity and simulation accuracy should be carefully considered for appropriate NoC designs.

2.4.5 Other Models and Tools for Specific NoC Simulations

Besides those general-purpose simulators, there are also several special-purpose models and tools for NoC system simulations or parametric measurement. **ORION** [87] [157] is a power and area model for high-level NoC designs. It is a popular model for performance estimation of NoC systems at early design stages. Its detailed high-level parametric abstractions of power cost in major network components help obtain accurate estimation, which leads to the model being integrated in many modern simulators. For example, the dedicated NoC simulator, NIRGAM, and the full-system simulator, GARNET, both use the Orion model for their power evaluation.

XpipesCompiler [158] is a tool to automatically instantiate application-specific NoCs for heterogeneous Multi-Processor SoC systems. Used in the particular range of heterogenous CMPs, this cycle-accurate tool is written in SystemC to support reliable, latency-sensitive operations on optimised network components. All the designed components in Xpipes are optimised to the custom communication needs of specific NoC architectures. The XpipesCompiler tool can emulate area, power and delay of NoC systems.

DSENT [159] (Design Space Exploration of Networks Tool) is a tool connecting emerging photonic components with existing electronic devices for modelling Opto-electronic NoCs. It is a unified framework that enables rapid design space exploration of cross-level network models and inherent interactions between photonic and electronic components. Such interactions impact on NoC system performance is quantified in this tool to emulate Opto-electronic NoCs in terms of timing, area and power metrics. The technology scaling, photonic parameters and thermal tuning of such NoCs are particularly modelled.

In this thesis, we focus on NoC designs cross a wide range. So the general-purpose simulators would be of higher utility for the research. Moreover, as our major objectives are exploring design space at the early design stages, balance between model accuracy and design productivity is of high concern, which spotlights the fitness of dedicated NoC simulators as the candidate platform for functional extensions. As per [33], the research areas of NoC design could be categorised into 4 domains: application modelling, network interconnection architecture analysis, network interconnection architecture evaluation and NoC design validation and synthesis. The NIRGAM simulator supports the realisation of first three domains, which shows the extensibility for requirements of high-level modelling: fast, sufficient accuracy and suitable complexity. Hence, NIRGAM is chosen as the candidate simulation platform used in this thesis for implementing experiments and extending new functions.

2.5 Summary

For designs of interconnection networks, large computations have become an increasingly demanding task. Parallel and distributed data processing have evolved rapidly, which introduces novel challenges in network characteristics that need to be tackled. Thus, the Network-on-Chip (NoC) related issues are firstly reviewed due to their critical role in future communication-centric system designs. Intolerably large design time and cost overheads in modern complex interconnection networks have boosted the fast development of high-level automation techniques to alleviate the design cycles of prototyping and time to market. This chapter secondly discussed system-level DA techniques for NoC designs with accurate and efficient modelling of highly concerned performance metrics. Moreover, partitioning various applications into computational tasks and mapping them onto various network architectures is significant to the overall system performance. A survey on current task-mapping techniques in NoC designs was thirdly conducted to investigate current progress for potential improvement. Finally, establishing a suitable framework to evaluate system-level models of NoC systems with a careful balance between accuracy and efficiency is an obvious open issue, such that a survey on modern simulators for system-level design automation of NoCs was given.

Chapter 3

System-Level Modelling of Networks on Chip

In this chapter, we firstly introduce the necessity of system-level design automation in complex NoC system designs. Then, the backbone simulator, NIRGAM, is introduced and the extended work based on it for most of our subsequent research is discussed. Next a case study is given to develop a simple one-to-one data transmission system using an asynchronous FIFO as channel port buffers in the NIRGAM simulator. The case study inspects the functional feasibility and accuracy of the NIRGAM simulator for high-level model abstraction and performance evaluation by comparing the power cost of the same system between its system-level behavioural model in the NIRGAM and gate-level design in the traditional synthesis design flow under a certain time period of transmission.

3.1 Necessity of High-level Model Abstraction to NoC

In traditional synthesis design flow of ASIC (Application-Specific Integrated Circuit) systems, functional behavioural models are firstly developed at RTL (Register-Transistor Level) and described by popular Hardware Description Languages (HDLs) like Verilog, SystemVerilog and VHDL based on specific design requirements. The HDL descriptions of system designs are validated their functionality by introducing a testbench to simulate the desirable behaviours. Then the functionally correct designs are synthesised to produce structural models. The generated gate-level netlist of designs are again validated with extra timing analysis, before being sent to the layout design stage for placing and routing on a chip. These steps so far are well-known as 'front end' design flow of ASIC systems. The 'Back-end' design flow normally starts from place and route phase. Once all modules of a complex system design are functionally validated and spatially placed, the design will be sent for manufacturing. The whole end-to-end process represents a popular design flow of digital electronic systems in Figure 3.1 [160].

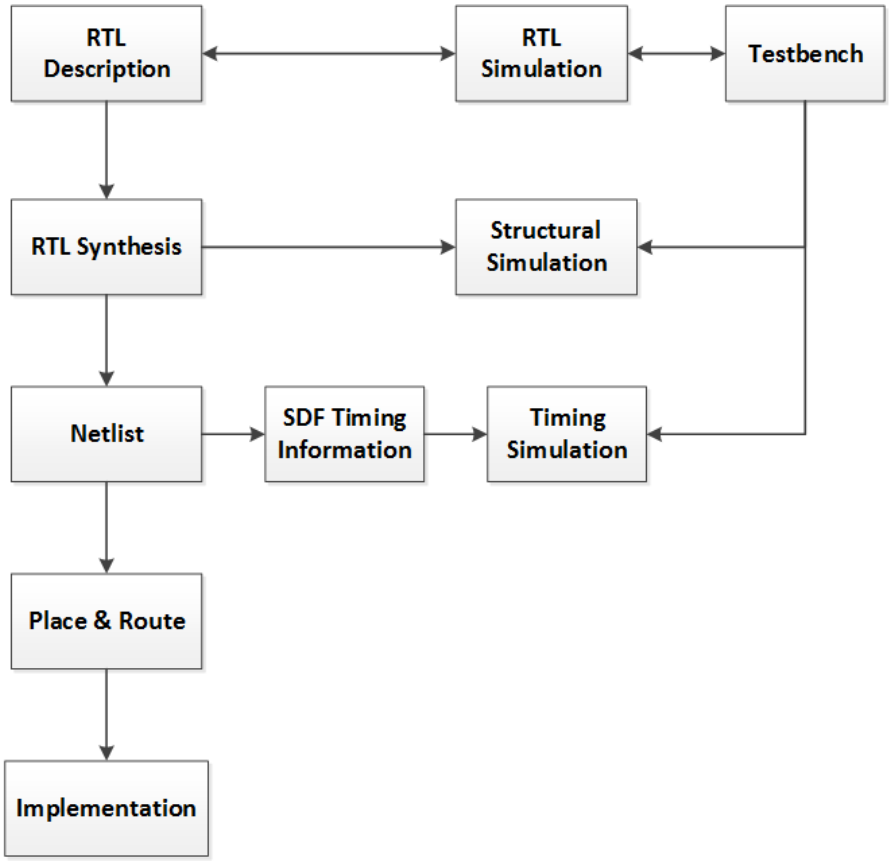


Figure 3.1: Basic Digital Design Process

For the design stages of simulating, validating and synthesising digital systems, as shown in the figure, many mature industrial tools like ModelSim [161], Design Compiler (DC) [162], Synopsys and Cadence products have been developed for years and widely used. Thanks to these tools, many steps in those design stages have been automatically implemented or a major part of design workloads has been undertaken, which greatly improves the design productivity. In particular, the place and route part was the first to be processed automatically, freeing developers and designers from the heavy repeat labour of layout drawing. Instead, they can spend more time on designing system specifications, which is the unavoidable and valuable stage that needs creativity.

However, this popular RTL design flow has faced a severe design challenge with the increasing complexity of on-chip systems that have more functional units integrated. The more modules that are integrated in a system, the more complex such a system will be. This increasing design complexity may lead to intolerably long design cycles for iterative module refinement and unacceptably heavy workloads of full manual prototyping, debugging and functional verification, which heavily exacerbate the design efficiency, productivity and time-to-market pressures.

In this case, figuring out new design methods becomes necessary to alleviate such a conflict between design complexity and design productivity. Developing more accurate system-level behavioural models of NoC architectures and functional modules using modern design automation techniques seems a promising candidate to accelerate the decision of proper design specifications and the simulation of desirable functional behaviours. It is necessary to validate the model accuracy to obtain intuitive perceptions of developing advanced and useful DA techniques. Specifically in this thesis, the backbone simulator, NIRGAM, is examined in terms of its model abstraction for NoC architectural and functional designs. Existing models and abstractions will be refined to meet the accuracy requirements. Moreover, extended models will be developed if certain functions or performance metrics are of a particular concern in the research.

3.2 Extended NIRGAM Simulator

3.2.1 Introduction to NIRGAM

Network-on-chip Interconnect Routing and Application Modelling (NIRGAM) [163] is a discrete-event, cycle-accurate simulator developed for Network on Chip research. It is written in SystemC with extensible modules. The NIRGAM simulator supports various user-designed topologies, routing algorithms and applications that are inserted into the simulator. Its performance evaluation of simulation results in terms of throughput, packet latency and power consumption can be generated explicitly by plotting graphs in Gnuplot or Matlab. Figure 3.2 shows the global architecture of the simulator used in this chapter.

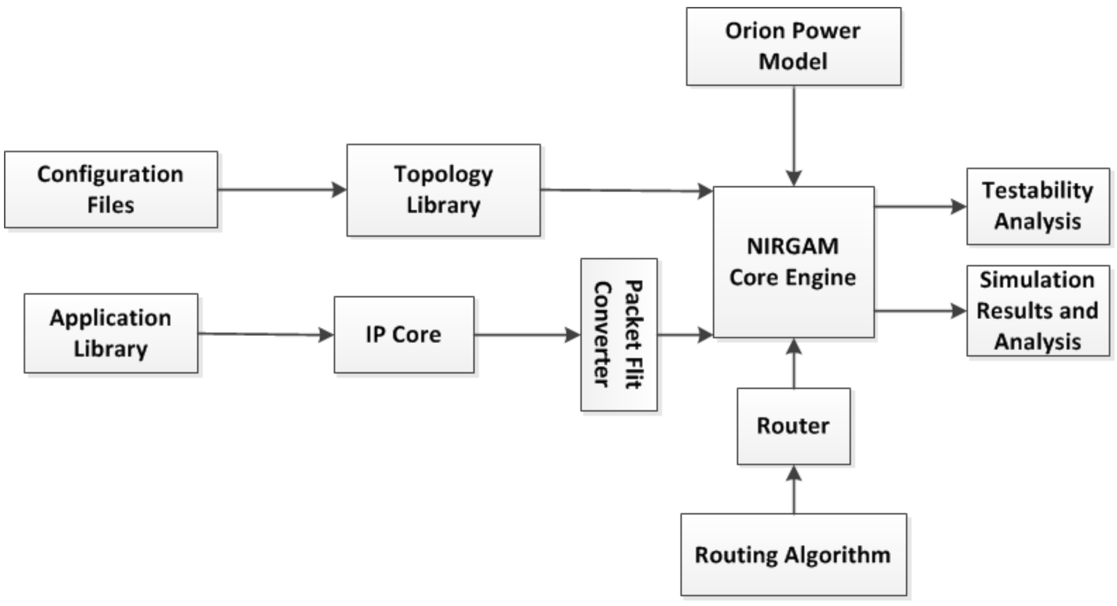


Figure 3.2: The System Architecture of NIRGAM Simulator

In the figure, configuration files set specific parameters like node attached applications, network topology, network size, simulation cycles and routing algorithms. The settings ensures specific network topology from available ones in the topology library and deliver it to the core engine module. The application library stores user-designed applications and functional units that will be attached to IP core models of network nodes. The attached applications are processed and implemented as IP cores when the network is established.

Once the processed data needs to communicate with other IP cores, the data packets will be sent to the packet flit converter module for further traversal across the network. The simulator uses wormhole switching flow-control method for data transmission. Each packet is split into arbitrary numbers of flits (flow control units) in which a head flit with destination information, intermediate flits with data and a tail flit are categorised. The converted data flits transmit across the network via certain router models based on specific patterns that are decided by the routing algorithm.

The core engine models networks with a number of homogeneous nodes interconnected in specific patterns. The node structure of regular networks is demonstrated on a typical 3 by 3 mesh topology shown in Figure 3.3. Each node in the network shares the same structure containing switch router connected by buffers and IP Processing core. The switcher includes input/output channel controllers, an arbiter, a virtual channel allocator and a crossbar.

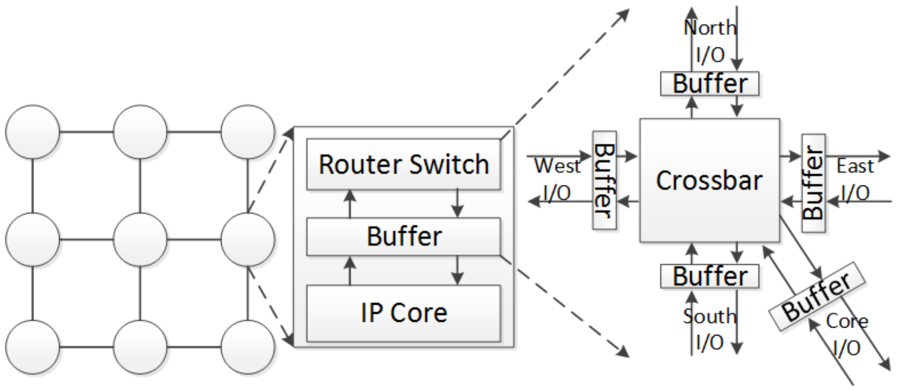


Figure 3.3: Typical Mesh Network with Homogeneous Node Architecture

In the router, input/output channel controllers manage all input/output channels that match four neighbour directions of the node plus one direction to the IP core. Buffers offer store-and-forward functions to incoming or self-generated flit data. If there is no processing request or data is being processed already by the IP core, the upcoming traversal direction of flit data will be arbitrated by the arbiter based on routing algorithms and system configuration. Then the crossbar allocates flit data to relative output channels. Both data receiving and sending processes depend on the system clock, which is also configurable.

If quality of service is expected to apply in some cases, the capacity of input/output channels can be altered by the virtual channel allocator to achieve a specific performance. The virtual channel allocator can divide physical channels into virtual channels to alter the channel bandwidth based upon the users' requirements. Once an NoC platform is established, various user designed applications can be implemented by the core engine and routed based on selected routing algorithms. NIRGAM can provide versatile performance metrics in terms of trace-based packet latency, network throughput and power consumption. In particular, trace-based timing analysis and the Orion power model are integrated to estimate the application performance in terms of timing and power.

3.2.2 Extended Work

The extended functional modules and modified models in the NIRGAM simulator for our research are shown in the red areas of Figure 3.4. The explanation of those modifications

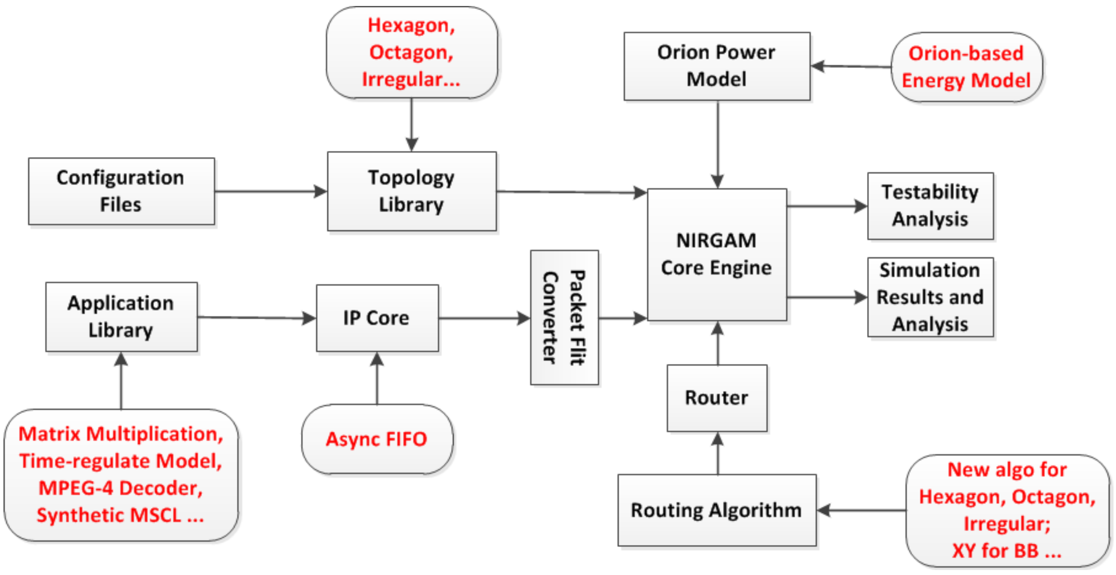

Hexagon, Octagon, Irregular...
Orion Power Model
Orion-based Energy Model
Configuration Files
Topology Library
NIRGAM Core Engine
Testability Analysis
Simulation Results and Analysis
Application Library
IP Core
Packet Flit Converter
Router
Matrix Multiplication, Time-regulate Model, MPEG-4 Decoder, Synthetic MSCL ...
Async FIFO
Routing Algorithm
New algo for Hexagon, Octagon, Irregular; XY for BB ...

Figure 3.4: The Extended NIRGAM Simulator

are listed below:

- **Asynchronous FIFO:**
  As per the research objectives given in Section 1.2 and Figure 1.2, we firstly explore the potential improvements in accurate analysis and estimates of system-level NoC modelling. The accuracy of abstract models in this step has direct performance influence on the design productivity of subsequent stages. To inspect the model accuracy at system level, a case study of designing an asynchronous dual-clocked FIFO as the buffer of router port channels in NoC nodes for data transmission is given. The FIFO structure is developed at both system level in the NIRGAM

simulator and at gate level in a synthesised design flow. The power model in NIRGAM, Orion, is also extended and refined based on parameters from an industrial cell library. Both system-level and gate-level FIFO designs are mutually compared to estimate the performance accuracy of NIRGAM models in terms of power and timing. The reason for designing asynchronous FIFO in NoC systems is also explained later in Section 3.3.1.

- **Orion-based Energy Model:**
  The energy performance metric is often paid more concern than the power metric in large-size or long-term simulations of NoC systems. It is because the energy is tied to specific tasks and the time required for those tasks, which represents the execution efficiency of systems for given tasks [164]. Thus, to better evaluate the performance of NoCs for specific applications, we have modified the original Orion power model in the NIRGAM simulator to offer energy estimates. In particular, power metrics of data transmitted across core buffers, router switches, channel ports, channel wires and physical links between node pairs in networks are transferred into energy metrics. Both dynamic switching and leakage energy of those network behaviours are included. In Chapter 5, we also add a new energy metric between the IP core and core buffers to construct a more precise energy model for performance prediction.

- **Time-regulated Model:**
  The time-regulated model is developed in the research of Chapter 4 for efficient topology emulation. As per the research objectives shown in Figure 1.2, system-level modelling of NoC architectures is significant to find optimised NoC platforms for specific applications. The architectural diversity and modelling efficiency of NoC systems are essential for high-efficiency searching of optimal NoC candidates, which improves the design productivity in subsequent stages. Hence, we explore the design space of the second research objective in Chapter 4, proposing an efficient method to emulate virtual NoC topologies in NIRGAM by attaching a time-regulated model to existing, reusable mesh nodes. More details of the model functionality and the emulation method will be discussed in that chapter.

- **New Non-rectangular and Irregular Topologies:**
  Several new topologies, including non-rectangular topologies (a honeycomb hexagon and a sparse-octagon) and relevant irregular topologies, are integrated in the topology library module of NIRGAM simulator in Figure 3.4. These new topologies are modelled at system level as a comparison for functional validation of virtual topologies emulated by our proposed method in that chapter. This functional extension is to provide design automation of more diverse NoC architectures at early design stages to improve existing network simulators. The reason for modelling non-rectangular NoC topologies is explained in Chapter 4.

- **New Routing Algorithms:**
  New deadlock-free routing algorithms specifically designed for given non-rectangular and irregular networks are developed in the routing algorithm module of NIRGAM simulator in Figure 3.4. This is to ensure fair performance comparisons of those NoC architectures with the virtual topologies emulated by our method proposed in Chapter 4. In addition, a modified XY routing algorithm is also developed in Chapter 5 to adapt to a modified branch and bound mapping algorithm proposed by our method in that chapter. Detailed explanation of these algorithms and the experiments they are used for are given in those two chapters, respectively.

- **Synthetic and Real-Media Applications:**
  In all the research of this thesis, it is required to evaluate many applications for functional validation and performance comparisons. Some of them are synthetic applications with simple structure and less complexity. The others are real-world applications with highly complex structure and contents that reflects realistic scenarios. These applications are designed and stored in the application library module of NIRGAM simulator in Figure 3.4. In later sections of this chapter, a pseudo-random number generator is developed to represent classic communication in NoC systems. In Chapter 4, both synthetic and real-media applications have been proposed for performance comparisons between different virtual and real network topologies along various routings. In Chapter 5, synthetic random benchmarks are further developed for comparative simulations and performance analysis. Details of these modifications will be introduced in each chapter.

3.3 Case Study: Asynchronous FIFO for NoC Buffer

In this section, comparative experiments between the synthesised gate-level design flow as shown in Figure 3.1 and system-level behaviour abstractions in the NIRGAM simulator are used as a case study to inspect the performance accuracy of high-level modelling in NIRGAM. A simple asynchronous FIFO has been developed in the NIRGAM simulator to extend the functionality. Based on it, the case study establishes a small one-to-one data transmission system, which consists of one sender, one buffer and one receiver, to represent a typical router interconnection in NoC systems. A long sequence of data traffic will be generated at one node and routed to the other. The performance and power costs for such data routings is evaluated in NIRGAM.

As a comparison, our case study also includes a similar data transmission system implemented by conventional RTL synthesis design flow at gate level. Verilog HDL is used for initial modelling and functional elaboration. After simulating its behaviours in ModelSim for functional verification, the designed system has been synthesised in Design Compiler to generate the netlist of structural models. The gate-level netlist is re-validated the functionality with timing simulations recorded. Power performance of the synthesised design has been evaluated using Synopsys PrimeTime (PT) tool with the recorded timing simulations. The same library for power analysis of gate-level design is used to extract the cell parameters that are imported into NIRGAM Orion power model. The power costs of both levels of simulations are compared to investigate the model accuracy of system-level abstractions in NIRGAM.

3.3.1 Necessity and Motivation

Most of the current NoCs are synchronous where all the network components are driven by one global clock. The synchronous NoCs are fast and area efficient. However, there are several **design challenges** in synchronous NoCs that researchers have discovered to be difficult to resolve.

- **Support for heterogeneous networks**
  Unlike most multi-processor chip networks in which each node is a homogeneous processor, an MP-SoC is generally a heterogeneous network where the different IP functional blocks have different functions and hardware structures. These IP blocks may be designed and validated with different clock frequencies, area sizes and driven by various working voltages. The IP design and validation differences increase the design complexity of network topology, compromise the latency performance of whole networks and make the timing closure of NoC system difficult [10].

- **Low power/energy consumption**
  As aforementioned, low power/energy cost is one of the most significant factors in modern NoC designs. Network power/energy consumption determines the maximum stand-by time of a mobile electronic device. The clock tree of synchronous NoC systems will consume a significant amount of energy. Thus, partially asynchronous or globally asynchronous locally synchronous (GALS) system structure [165] may be one potential solution to mitigate the energy cost, which requires asynchronous component designs in NoC systems.

- **Tolerance to variation**
  In deep sub-micron/nano VLSI designs, variations of process, temperature and voltage become significant impact factors on the performance of digital systems. The variation-caused delay uncertainty in sign-off timing closure will be more than 30% in 2024 according to the analysis of the International Technology Roadmap for Semiconductors [166]. Hence, traditional static timing analysis will be replaced by statistical timing analysis methods to tackle dropping yield rates and over-conservative timing estimates. Synchronous on-chip networks can mitigate such uncertainty by considering variations among the task mapping

procedure. However, this may work only in homogeneous networks instead of all network structures and the routers may have to work at the worst-case speed.

To tackle these design challenges, asynchronous functional design in NoCs may be a useful proposal especially in the communication part of NoC systems. Such designs are built with clock-less asynchronous circuits using handshake protocols, which are insensitive to delay. The interface between IP blocks and the network can be unified by the same synchronous to/from asynchronous interface due to such delay insensitivity. All synchronous blocks will be isolated by the interface, which enables substantially simpler chip-level timing closure. Furthermore, the asynchronous design components are intrinsically tolerant to variations because the communicating handshake protocols they use are delay-insensitive. Besides, zero dynamic power will be consumed in asynchronous NoC components if there is no data transmission.

Since asynchronous designs in NoC components are naturally slower and more area-consuming than the synchronous NoC components that have similar structures, potential asynchronous designs in high-level extraction simulators need to be carefully considered for accurate performance estimates. Based on the characteristics of asynchronous components, an asynchronous FIFO as the input/output buffer in NoC router structures is a good starting point. Such a FIFO mechanism is one of the most efficient and widely used methods to synchronise the communication interface of a GALS NoC [167]. An simple FIFO structure can largely mitigate the speed and area overheads, which may less degrade the performance.

To our best knowledge, few existing simulators have supported system-level modelling and performance simulations of asynchronous components. To fill this gap, we are motivated to implement a system-level dual-clocked FIFO model in the NIRGAM simulator as the input/output buffers of node routers for design automation of NoC architectures. More detailed explanations are given in the subsequent sections.

3.3.2 Asynchronous FIFO Structure

Normally, an asynchronous FIFO indicates a FIFO structural design that has the data values written into it in one clock domain, then read from it in another clock domain. The two basic clock domains are asynchronous with respect to each other. This structure is commonly used in many digital IC designs due to its structural simplicity. The proposed asynchronous FIFO component for NoC design is developed from the design structure introduced in [16]. As shown in Figure 3.5, the FIFO implements asynchronous transmission by using Gray code counters as write and read pointers, comparing them to generate asynchronous control signal for full and empty signal indications.

The full and empty signal generating process is of the highest significance for asynchronous transmission as well as buffer fetch-and-store activities. Besides, the

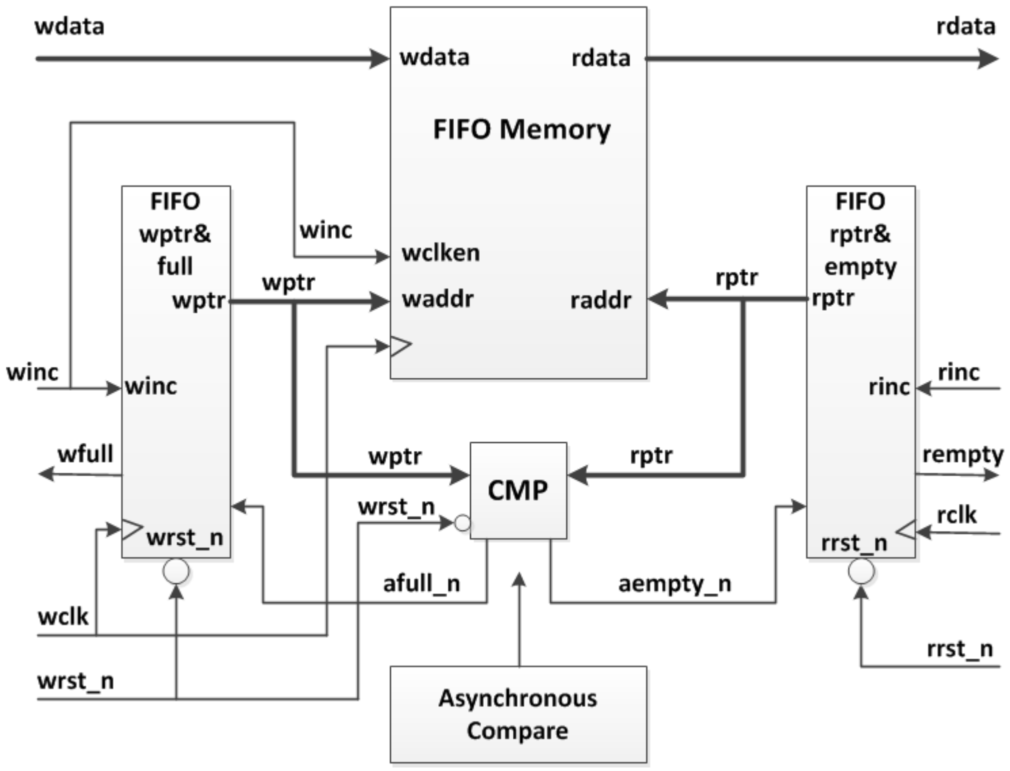


Figure 3.5: The Applied Asynchronous FIFO Structure [16]

correct implementation of the full and empty signals are usually the hardest design part. The traditional way to correctly implement full and empty signals is to append one extra bit onto both pointers and to compare them. The approach used in [16] gives a improved solution with less area. It compares the write and read signals stemming from two different asynchronous clocks in the comparator unit (shown as CMP in the figure), dividing the address into four quadrants and determining full or empty status according to the two MSB (Most Significant Bit) of the counters.

Moreover, the Gray count sequence only changes one bit at one time in order to eliminate potentially unreliable decoding spikes caused by simultaneous changes on more than one bit of two asynchronous pointers. The drawback of such modifications is the cost of extra clock cycles to execute all the operations.

3.3.3 FIFO High-Level Modelling

In this chapter, we test the behavioural and performance accuracy of system-level design automation by comparing with gate-level synthesised design. Specifically, this is achieved by implementing the asynchronous FIFO using both the RTL synthesis design flow and high-level functional modelling in the NIRGAM simulator. For high-level design automation, the asynchronous FIFO is designed in NIRGAM. More precisely, we simply develop the asynchronous FIFO structure as a functional extension of network node buffers to replace the default ones, such that the asynchronous data reading and writing processes are constructed when applications are processed and the NoC topology has been built up. The asynchronous read and write signals operate based on different clock domains but both are extracted from the same global clocks.

In NIRGAM, the asynchronous FIFO is used as the input/output channel buffer of node routers. As shown in Figure 3.6, the network node and router structures are given in black. Components in red are where the replacement occurs. A normal channel buffer is replaced by the asynchronous FIFO, providing two clock domains for its two buffer ports. The operations at the port side that connect to the local node router run under clock domain 1, which can be asynchronous to clock domain 2 which operates the other buffer port that connect to the eastern neighbouring node router.

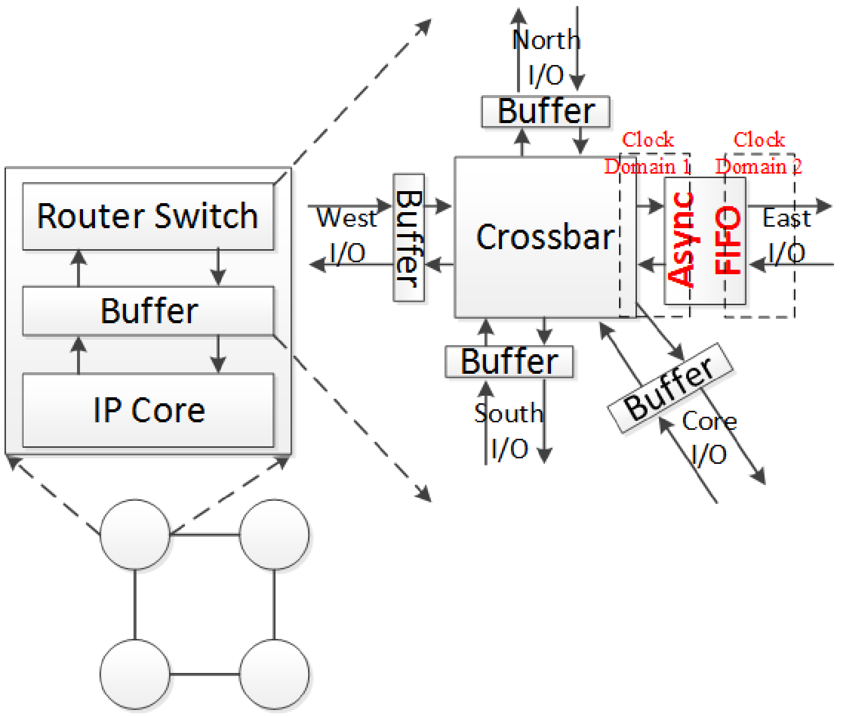


Figure 3.6: System-Level Modelling of Asynchronous FIFO as NoC router Channel buffers in NIRGAM

3.3.4 FIFO Gate-Level Implementation

As per the current popular RTL synthesis design process, we initially design the asynchronous FIFO structure at RTL using Verilog HDL [16]. The functionality of behavioural FIFO models are validated by writing and simulating a proper test bench in the ModelSim simulator. The validated design is then synthesised in Design Compiler (DC). This gate-level design is re-validated its functionality by simulating again in ModelSim. With the time delay information of the netlist, stored as a Standard Delay Format file (.sdf file), the FIFO design generates VCD (Value Change Dump) file of the simulation. The VCD file with its timing simulation and the netlist file with its structural description are imported into Synopsys PrimeTime (PT) for power analysis. The whole design, synthesis and implementation flow is elaborated in Figure 3.7.

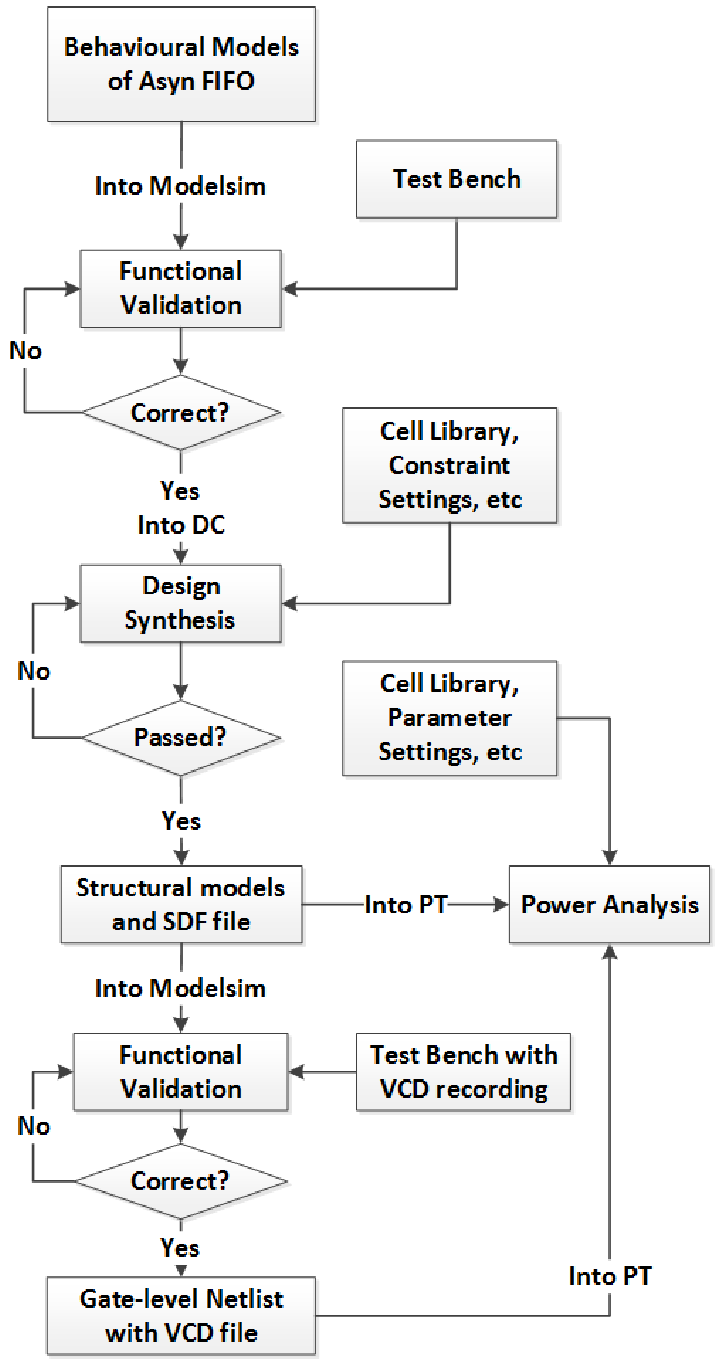

Behavioural Models of Asyn FIFO
Into Modelsim
Test Bench
Functional Validation
No
Correct?
Yes
Into DC
Cell Library, Constraint Settings, etc
Design Synthesis
No
Passed?
Yes
Cell Library, Parameter Settings, etc
Structural models and SDF file
Into PT
Power Analysis
Into Modelsim
Functional Validation
Test Bench with VCD recording
No
Correct?
Yes
Gate-level Netlist with VCD file
Into PT

Figure 3.7: Gate-Level Synthesis and Power Analysis Flow of Asynchronous FIFO

3.4 Model Accuracy Analysis

In this section, the performance of the system-level system in NIRGAM is compared with the gate-level design. In Subsection 3.4.1, functionality of the given asynchronous FIFO is validated at gate level. In Subsection 3.4.2, a one-to-one transmission system using this asynchronous FIFO is designed at both levels. Their performance are compared to inspect the accuracy of system-level models.

3.4.1 Gate-level Asynchronous FIFO

3.4.1.1 Experimental Setup

In this subsection, the synthesisable gate-level FIFO structure given in Figure 3.5 is verified for proper data storage and fetching under different clock cycles. A test bench is developed for the functional validation. The asynchronous FIFO at both gate level and system level has 32-bit data size and 8-bit address size. The FIFO memory is 256 $\times$ 32 (Memory Depth $\times$ Memory Width, the memory has $2^{addrsize} = 2^8 = 256$ entries and each entry is 32-bit wide). The proposed gate-level designs, including FIFO and transmission system, are all implemented by using the cell library of TSMC (Taiwan Semiconductor Manufacturing Company) 90-nm technology.

As per [10], asynchronous designs often cost more area and operates at lower speed in NoC designs. The operating frequencies applied in state-of-the-art asynchronous FIFOs for GALS NoC routers are around 340 Mhz [167] to 1.13 Ghz [168]. Hence, the write clock frequency (wclk in Figure 3.5) for both levels of designs in our case are all set to 1 Ghz (1 ns/cycle). In the gate-level FIFO, the read clock frequency is set to 625 Mhz (1.6 ns/cycle). The read clock is initiated a little later (about 0.3 cycle delay) than the write clock, which enabled staggered writing and reading behaviours to be observed. Clock jitter is also designed in the test bench for the rclk signal. The jitter insertion is set in test bench file to have floating small values ranging from -0.2 to $+0.2$ ns/cycle adding to each cycle of read clock, making its values between 1.4 ns/clock to 1.8 ns/cycle. The verilog codes for inserting the jitter to the clock signal is shown in Figure 3.8

The warm-up simulation period in both gate-level and system-level designs is set to 5 cycles. The total length of simulation time period is $100,000$ cycles. After the warm-up period till the end of simulations, a pseudo-random number generator is given in the test bench to feed continuous random unsigned numbers with 32-bit width and ranging from 0 to $2^{32} - 1$ as the input data. This traffic represents common network communication behaviours in NoC systems. The functions of the asynchronous FIFO for validation in our case include correct data transmission under different clock domains and/or jitter insertion, correct stop reading/writing data from/to FIFO memory when FIFO address is empty/full. All situations are tested under post-synthesis netlist simulations.

```verilog
`timescale 100ps / 1ps  //  time_unit/time_precision

//define the time scales
//wclk: 1000Mhz = 1G = 1 ns/cycle && rclk: 625Mhz = 0.625G = 1.6 ns/cycle
`define WCLK 10
`define RCLK 16 //with jitter

//rclk with jitter
  initial
    begin
      rclk = 1'b0;
      forever
        begin
          //give jitter values between -0.1 and 0.1 ns per half cycle
          rclk = 1'b0;
          #(`RCLK/2 + $dist_uniform(seed,0,1));
          rclk = 1'b1;
          #(`RCLK/2 + $dist_uniform(seed,0,1));
        end
    end
```

Figure 3.8: Verilog Code for Jitter Insertion in Test Bench

3.4.1.2 Result Analysis

For functional validation, the asynchronous FIFO is firstly simulated for basic, normal write and read operations in Figure 3.9. Arrows in this figure, as well as in subsequent figures in this section, indicate each relevant signal of the simulations, rather than one specific number of the sequence. From the arrow indicators, it is seen that a sequence of random numbers is fed into the FIFO as inputs on each write clock cycle. Once the read enable signal goes high, the FIFO starts to output the stored numbers based on the read clock cycles. The write and read clock signals are of the same frequency but fed to the FIFO with staggered time stamps. This netlist simulation result shows the correct functionality of an asynchronous FIFO.

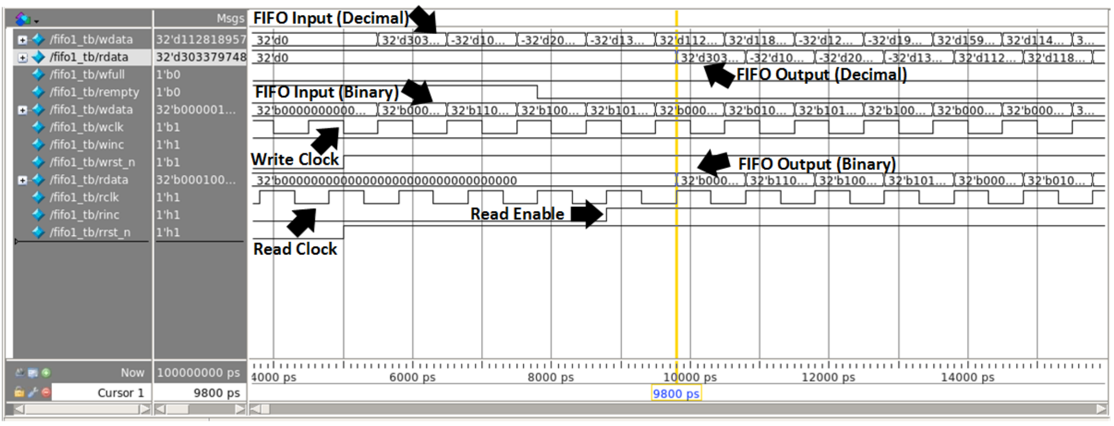
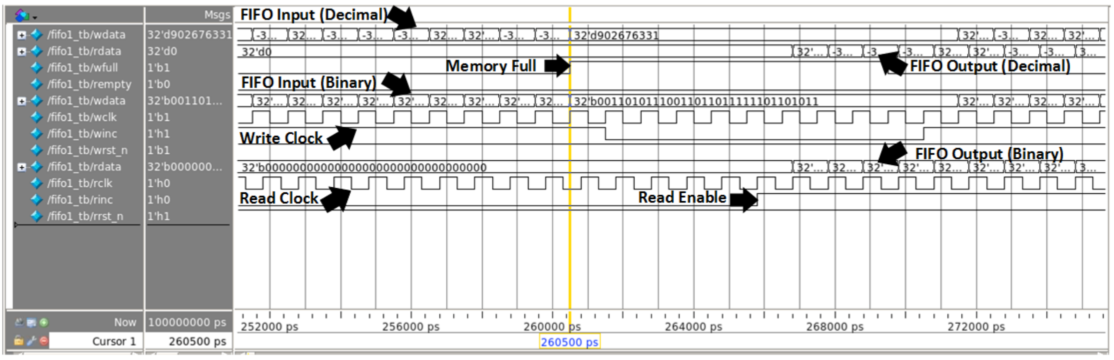


Figure 3.9: Functional Verification of Asynchronous FIFO

Without changing clock settings, we set the test bench to let the asynchronous FIFO only write data into the memory until the addresses are fully occupied. Then the read operation is allowed to output data from FIFO memory. Figure 3.10 shows the correct behaviours of the asynchronous FIFO under this scenario. Those arrows shown in the figure represent signals that are used in this simulation. It is observed that the write full signal goes high when the memory is full. After that, the FIFO stops writing data into the memory. Once the read enable signal goes high, the FIFO starts to output the stored data. When the memory addresses are released due to the output operations, the write full signal goes low again and the generated random numbers continues to be input into the FIFO.

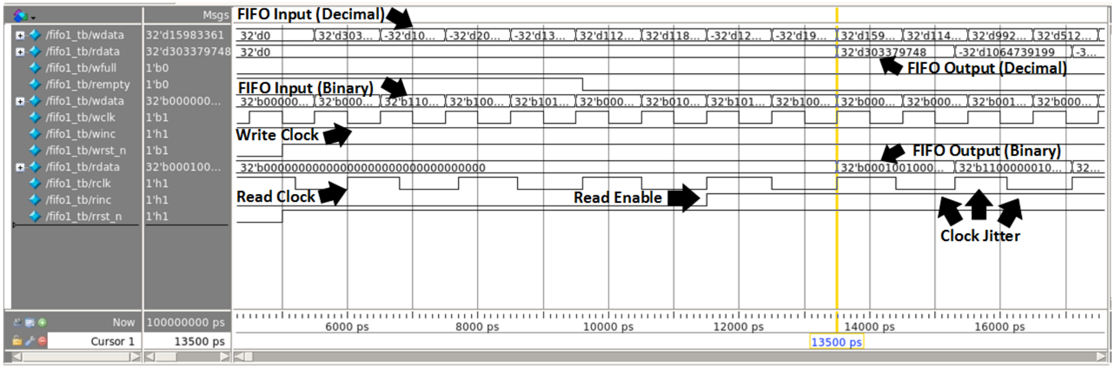


```
                        Msgs
/fifo1_tb/wdata     32'd902676331
/fifo1_tb/rdata     32'd0
/fifo1_tb/wfull     1'b1
/fifo1_tb/rempty    1'b0
/fifo1_tb/wdata     32'b001101...
/fifo1_tb/wclk      1'b1
/fifo1_tb/winc      1'h1
/fifo1_tb/wrst_n    1'b1
/fifo1_tb/rdata     32'b000000...
/fifo1_tb/rclk      1'h0
/fifo1_tb/rinc      1'h0
/fifo1_tb/rrst_n    1'h1

Now       100000000 ps
Cursor 1  260500 ps
```

Figure 3.10: Functional Verification of Asynchronous FIFO with full memory

To test the FIFO functionality with jitter insertion, different write and read clocks (1 ns/cycle and 1.6 ns/cycle) are configured. The jitter is inserted at the read clock signal as Figure 3.8 shows. The simulation result is given in Figure 3.11 with arrows indicating each signal. As illustrated, the fed data inputs are successfully stored in the FIFO memory and correctly output under separated clock signals. It is observable that the write and read clocks are not only fed at staggered time stamps but also with different cycle length. The clock jitter arrow indicator presents the unequal length between a pair of read clock cycles, suggesting the successful jitter insertion. As the write and read operations function correctly, the FIFO under asynchronous clocks with jitter insertion is validated.

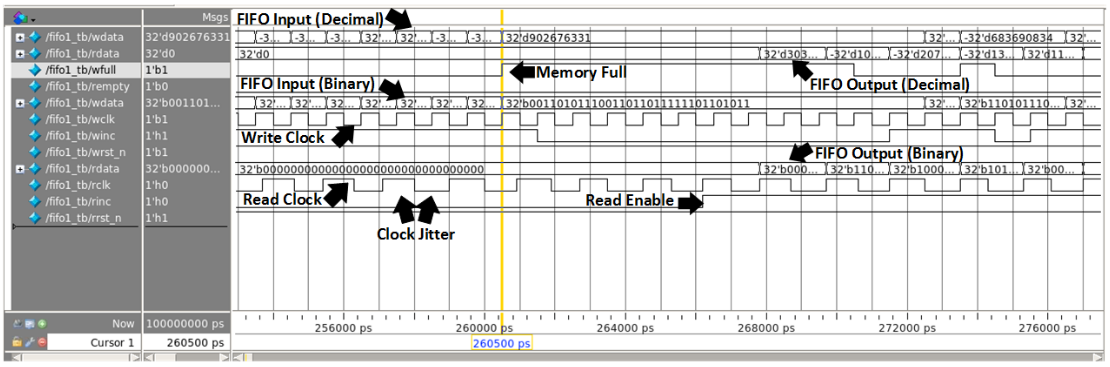


```
                        Msgs
/fifo1_tb/wdata     32'd15983361
/fifo1_tb/rdata     32'd303379748
/fifo1_tb/wfull     1'b0
/fifo1_tb/rempty    1'b0
/fifo1_tb/wdata     32'b000000...
/fifo1_tb/wclk      1'b1
/fifo1_tb/winc      1'h1
/fifo1_tb/wrst_n    1'b1
/fifo1_tb/rdata     32'b000100...
/fifo1_tb/rclk      1'h1
/fifo1_tb/rinc      1'h1
/fifo1_tb/rrst_n    1'h1

Now       100000000 ps
Cursor 1  13500 ps
```

Figure 3.11: Functional Verification of Asynchronous FIFO with Clock Jitter

Figure 3.12 shows the functional validation of the asynchronous FIFO with full memory and jitter insertion. In the figure, the clock jitter is again inserted into the read clock signal. The memory full signal goes high and the FIFO stops writing when the FIFO memory is full with the continuous random numbers. Then the read enable signal is set to high so that the data stream is output sequentially to release the memory addresses. As shown, the FIFO starts to input data and the memory full signal goes low after the read operations start. All the mentioned signals are given by the arrows in the figure. This process validates the expected functions achieved by the asynchronous FIFO.


```
                     Msgs
/fifo1_tb/wdata      32'd902676331
/fifo1_tb/rdata      32'd0
/fifo1_tb/wfull      1'b1
/fifo1_tb/rempty     1'b0
/fifo1_tb/wdata      32'b001101...
/fifo1_tb/wclk       1'b1
/fifo1_tb/winc       1'h1
/fifo1_tb/wrst_n     1'b1
/fifo1_tb/rdata      32'b000000...
/fifo1_tb/rclk       1'h0
/fifo1_tb/rinc       1'h0
/fifo1_tb/rrst_n     1'h1

Now                  100000000 ps
Cursor 1             260500 ps
```

Figure 3.12: Functional Verification of Asynchronous FIFO with Clock Jitter and Full Memory

From the above simulations, it is notable that the designed gate-level asynchronous FIFO performs correctly under various circumstances. In particular, the clock jitter has not disturbed the performance. This feature is useful for comparing to the system-level design because the FIFO in NIRGAM has no clock jitter model. Hence, to obtain a fair performance comparison, no clock jitter is inserted in both system-level and gate-level system designs given in the next subsection. Similarly, NIRGAM has different clock signals but both are derived from the same global clock, which results in the same cycles for both clocks. Therefore, the write and read clocks of the one-to-one system designed at both levels in the next subsection have equal cycles and the same staggered initialisations for the fair comparison.

3.4.2 Data Transmission System

3.4.2.1 Experimental Setup

In this subsection, one-to-one transmission system using the given asynchronous FIFO is designed at both gate level and system level for performance comparison. For the system-level design automation and modelling in NIRGAM, the transmission system is used with the asynchronous FIFO as one router port buffer. A 2-by-2 2D-mesh network, which is the smallest network available in NIRGAM, is given in Figure 3.13. A sender and a receiver of the asynchronous system are implemented onto specific network nodes

($R0$ and $R2$) as attached applications. The other two nodes are used just for the topology formation and thus excluded from the experiment. Each network node consists of an IP core and a router that connects to other nodes by linking channel ports on their routers. Each router has five directional ports for data communication in the network and each port has a buffer constructed by a synchronous FIFO. The FIFO size used for port buffers in NIRGAM NoCs are the same as the RTL asynchronous FIFO used in the last subsection (32-bit data size, 8-bit address size, 256-bit FIFO memory depth).

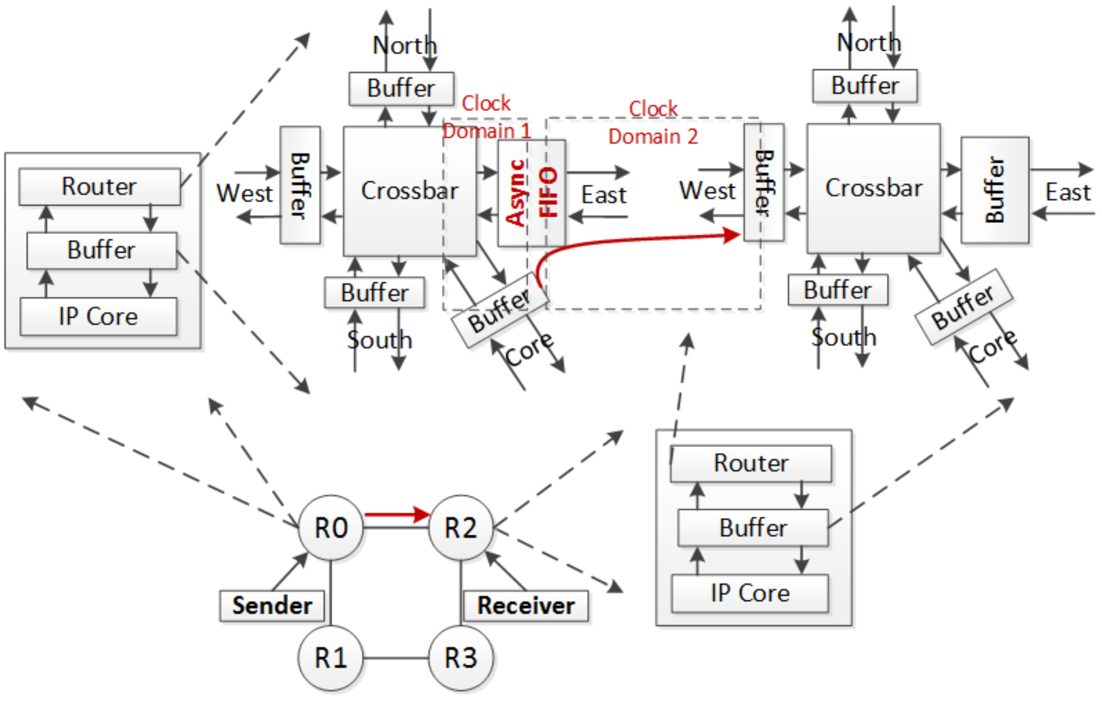


Figure 3.13: High-Level Asynchronous Transmission System designed in NIRGAM

In our comparative experiment, the eastern port buffer of router on the sender node $R0$ is replaced with the proposed asynchronous FIFO. A pseudo-random number generator with the same settings (32 bits/flit, 1 cycle/flit) as in the last subsection is modelled in the IP core of the sender node. Its generated data stream is stored to the core buffer under the write clock (clock domain 1, the same as the input clock of asynchronous buffer in Figure 3.13) for feeding a data sequence to the asynchronous port buffer as the traffic flow. The simulated traffic transmission process in the system-level asynchronous system is shown as the red line in Figure 3.13. Once the random numbers are fed as data flits to the asynchronous port buffer, the FIFO memory stores data under the write clock and fetches data from the memory under the read clock (clock domain 2). Data flits out of the asynchronous port buffer leave the sender node $R0$ and are routed to the input port at the receiver node $R2$. Under the same read clock, the input port buffer synchronously stores the incoming data to its memory. Hence, this transmission process separates the working frequencies for network communication part ($R0$ node router to $R2$ node router) and in-node processing part ($R0$ IP core to $R0$ router), performing asynchronous data transmission across different functional blocks. The power cost of the asynchronous FIFO at the $R0$ router is measured. Specifically, the transmission behaviours of sending data from the $R0$ core buffer to the asynchronous FIFO, and outputting data from the asynchronous FIFO at $R0$ router buffer to the $R2$ router buffer are exercised.

As mentioned at the end of the last section, the clock settings in the transmission system designed at both levels are the same. (It is noted that the the clock settings for the transmission systems designed at both levels in this subsection are different from the settings of the gate-level asynchronous FIFO given in the last subsection.) So in the high-level design, both write and read clocks are set to 1 ns/cycle with about 0.3 ns staggered initialisation time stamps for clear observation. The total simulation time lasts $100,000$ cycles with the first 5 cycles as warm-up period, which is the same as the RTL simulations. The power performance of the data transmission is simulated by the integrated Orion power model [87] [157]. All the high-level models of the asynchronous system in NIRGAM for power performance emulations are based on the same cell library used for the RTL asynchronous FIFO and transmission system designs (that is, TSMC-90nm library). We have extracted the cell parameters of the library to replace the original settings in Orion models to ensure the accurate measurement of power performance at system level. In our experiments, the power measurement of the entire transmission process in NIRGAM starts from the generated random numbers inputting to the asynchronous port buffer of the sender node router as the data transmission, and ends at the storage of the final data of random numbers at the port buffer memory of the receiver node router. The average power cost of transmitting those random numbers is simulated for result comparison.

In a gate-level asynchronous data transmission system, one sender, one receiver and one asynchronous FIFO as their buffer are involved for data communications. The structure of the whole system is depicted in Figure 3.14. Design processes of synthesis, simulation and power analysis to this gate-level system is implemented according to the design flow given in Figure 3.7. All the simulation configuration and settings are the same as the high-level system design in NIRGAM. The power cost of data traffic simulation is reported in PrimeTime.

As shown in the figure, the sender and FIFO are in one clock domain, which is write clock (wclk), while the receiver is in the other clock domain, which is read clock (rclk). Since the power cost of generating a data stream at IP core and transmitting to the core buffer in the system-level design is not measured in NIRGAM, the gate-level asynchronous transmission system should also exclude the hardware of random number generator to maintain the consistency. Hence, as in the FIFO experiment in Subsection 3.4.1, the data of random numbers comes from the test bench. The whole process where the average power cost of transmitting all the generated random numbers is measured at gate level consists of sending the random numbers from the sender to the asynchronous FIFO (corresponding to the data sending from core buffer to asynchronous port buffer of

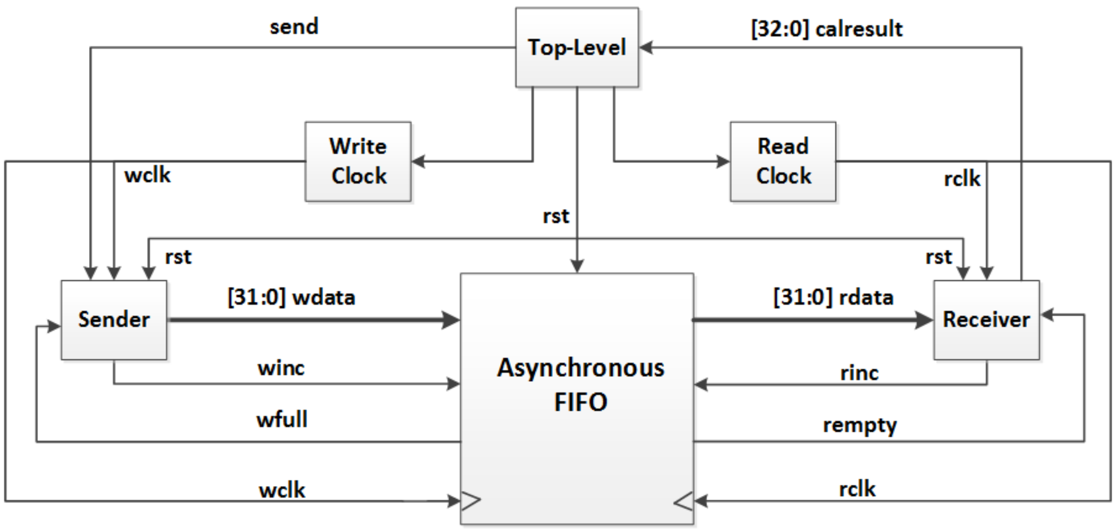


Figure 3.14: Gate-Level Asynchronous Transmission System for Performance Comparison

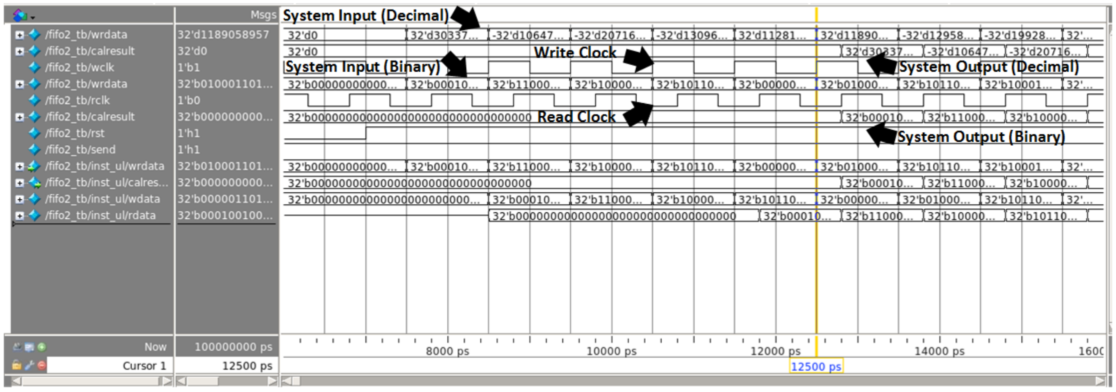
sender node in the high-level design), fetching data from the asynchronous FIFO memory to the receiver (corresponding to the data output from sender port buffer to the input port buffer of receiver node in the NIRGAM design) and receiving these numbers at the receiver (corresponding to the data stored to the input port buffer of receiver node in the NIRGAM design). This transmission process is similar to the simulation in the high-level NoC design for measuring the average power costs.

3.4.2.2 Result Analysis

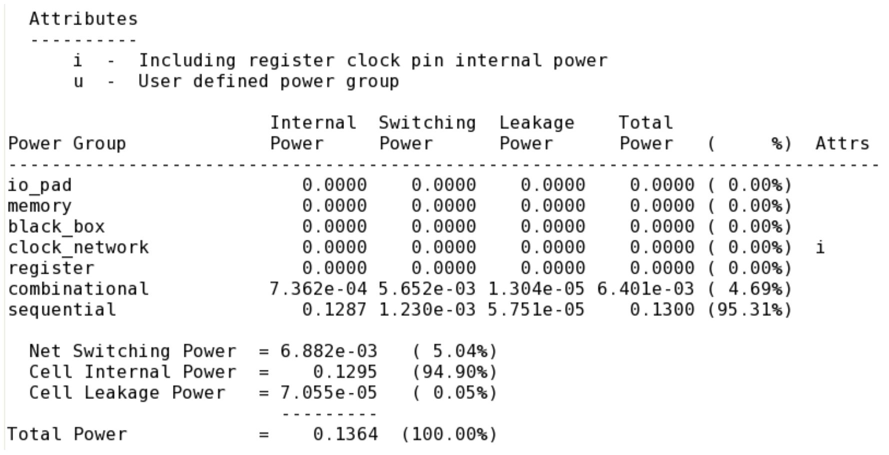
Figure 3.15 demonstrates experimental results of the gate-level synthesised asynchronous transmission system. The functionality of the system after synthesis is validated in Figure 3.15(a). Arrows in the figure indicate all the signals used in the experiment. As shown, the system can correctly input a continuous number sequence fed from test bench under the write clock and output them from the system under the read clock. It is seen that the whole simulation process lasts $100,000$ cycles with two clocks both at frequency 1 Ghz and staggered by 0.3 cycle. No jitter is inserted. The power cost by this gate-level system during such a simulation process is shown in Figure 3.15(b). The result unit is Watt and the report is generated in PrimeTime after a power analysis.

For the NIRGAM high-level asynchronous system shown in Figure 3.13, a similar simulation process is implemented based on the same settings as the gate-level system. The power performance of the system-level system is measured by the modified Orion power model. The performance result is generated and automatically saved in the simulator, which is illustrated in Figure 3.16.

To explicitly analyse the performance comparison, Table 3.1 lists the power results of the asynchronous transmission system designed at both levels. It is observable that both


Msgs
System Input (Decimal)
Write Clock
System Input (Binary)
System Output (Decimal)
Read Clock
System Output (Binary)
/fifo2_tb/wrdata 32'd1189058957
/fifo2_tb/calresult 32'd0
/fifo2_tb/wclk 1'b1
/fifo2_tb/wrdata 32'b010001101...
/fifo2_tb/rclk 1'b0
/fifo2_tb/calresult 32'b000000000...
/fifo2_tb/rst 1'h1
/fifo2_tb/send 1'h1
/fifo2_tb/inst_ul/wrdata 32'b010001101...
/fifo2_tb/inst_ul/calres... 32'b000000000...
/fifo2_tb/inst_ul/wdata 32'b000001101...
/fifo2_tb/inst_ul/rdata 32'b000100100...
Now 100000000 ps
Cursor 1 12500 ps
8000 ps
10000 ps
12000 ps
14000 ps
12500 ps

(a) Functional Verification of Gate-Level Asynchronous Transmission System



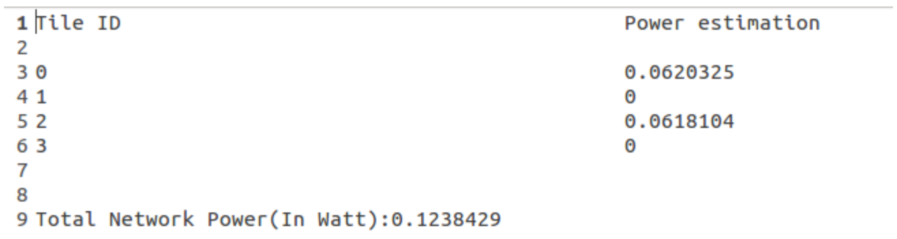
```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)
clock_network              0.0000    0.0000    0.0000    0.0000 ( 0.00%)  i
register                   0.0000    0.0000    0.0000    0.0000 ( 0.00%)
combinational           7.362e-04 5.652e-03 1.304e-05 6.401e-03 ( 4.69%)
sequential                 0.1287 1.230e-03 5.751e-05    0.1300 (95.31%)

  Net Switching Power  = 6.882e-03   ( 5.04%)
  Cell Internal Power  =    0.1295   (94.90%)
  Cell Leakage Power   = 7.055e-05   ( 0.05%)
                         ---------
Total Power            =    0.1364  (100.00%)
```

(b) Power Cost of Gate-Level Asynchronous Transmission System in PrimeTime

Figure 3.15: Functional Verification and Power Cost of Gate-Level Asynchronous Transmission System



```
Tile ID                                          Power estimation

0                                                0.0620325
1                                                0
2                                                0.0618104
3                                                0


Total Network Power(In Watt):0.1238429
```

Figure 3.16: Power Cost of System-Level Asynchronous Transmission System in NIRGAM

results are so close that less than 10% error is achieved. More specific, the power simulation of high-level system design in NIRGAM is merely 9.24% short in result measurement with respect to the power analysis of RTL synthesis system, indicating high model accuracy in the NIRGAM simulator for automating the asynchronous transmission at system level.

	Total Power (W)	Result Error
System-Level Design in NIRGAM	0.1238	9.24%
Synthesised Gate-Level Design	0.1364	

Table 3.1: Comparison of Power Costs of Both Levels of Designs for 100,000-cycle Data Transmission

3.4.3 Result Analysis of Case Study

By launching this case study, several conclusions can be drawn as follows:

1. Compared the simulation results of system-level modelling in NIRGAM to the RTL design, the performance error in terms of power cost along a long-term experimental time is small, indicating high accuracy of the system-level models in NIRGAM. This high accuracy of high-level models comes from the thorough abstraction of the system behaviours and structures. High consistency of the modified Orion model and the RTL cell library also contributes to this accurate comparison. However, although most of the cell parameters used in Orion model (cell width, length, capacitance, transistor types etc) have been unified with the RTL cell library, some cell settings in the library (parametric settings under various temperatures and noises) are still unmatched within the Orion model, which causes this trivial error in simulation results. Besides, since the average power costs of transmitting random numbers are measured in the simulations at both levels, the slight difference of total switching activities achieved in the high-level NIRGAM simulation and the gate-level simulation also impact on the error between their measured results. A potential solution to those potential reasons that cause the result error is extracting more parametric and environmental settings into high-level models.

2. In the FIFO experiments of Subsection 3.4.1, it is observed the clock jitter insertion has not disturbed the functional performance. But the power performance may vary since many components of an on-chip network (like crystal oscillator, PLL (Phase-Locked Loop), clock buffers, wire coupling and so on) may cause clock jitter. But it is unfortunate that NIRGAM cannot implement clock jitter since the high-level model of relevant components like crystal oscillator is absent in the simulator. This lack makes it unable to measure the influence of clock jitter to the performance of system-level designs. A possible solution is to improve the high-level automation in NIRGAM with extra models that includes description/abstraction of more realistic scenarios.

3. The asynchronous FIFO structure used in our case study is derived from previous work in [16], which is a good starting point to develop high-level models of GALS NoCs. Yet it is also witnessed in our case that larger memory space is allocated to

the asynchronous buffer rather than synchronous ones in NIRGAM, which is one trade off for the asynchronous function. Recent research [167] has suggested that the large area cost of the FIFO memory may make this kind of FIFO start to be less suitable as network routers due to the tightening area budget in modern NoC designs. Advanced asynchronous FIFO structures may need to investigate. Hence, finding more suitable asynchronous NoC components and accurately modelling their system-level design automation would be the further research.

3.5 Summary

For the contributions of this chapter to the thesis, we have extended the functionality of the NIRGAM NoC simulator. The performance estimate models in NIRGAM are modified to enable evaluations of either power or energy performance. A new FIFO functional module is developed to enable an asynchronous buffer at specific network nodes for system-level modelling. Based on the asynchronous FIFO, a case study of asynchronous data transmission system has been built up at both system level in NIRGAM and gate level. The performance simulations in terms of power cost within a certain time period are compared to validate the accuracy of system-level models. The Orion power model integrated in NIRGAM has been modified with the cell parameters derived from RTL synthesis technology library to calibrate the performance measurement.

The power result of the asynchronous transmission system simulated by system-level modelling in NIRGAM is close (less than 10% error) to the gate-level post-synthesis system, indicating high accuracy of our system-level models developed in the simulator. Through the error in our case is small, its possible solution is also discussed, which suggests two potential directions for improving the high-level design automation:

- refining the abstract models with higher accuracy and more detailed features;
- and constructing more generalised and representative models that cover more realistic scenarios.

These two potential directions have underpinned our ongoing research in subsequent chapters. Specifically, the research in Chapter 4 is based on the first direction and the work in Chapter 5 is based on the second direction.

Chapter 4

Efficient Modelling of Non-rectangular Topologies

A proper network topology has great influence on system performance [3]. For instance, hexagonal topology is used in cellular networks for maximal coverage area of communication, and irregular topologies are used in Wireless Sensor Networks to make full use of the 3-dimensional space. In designing these different kinds of networks, finding proper network architectures to specific applications is usually an important stage to optimise since the total number of design cycles can be reduced if proper network topologies can be quickly found and accurately verified at system level. Thus, efficiently simulating the performance of different network topologies at system level could facilitate the search of a proper network for a specific application. However, most current high-level simulators only provide performance simulations to Manhattan-based topologies due to the easy implementation. It is difficult to automate non-Manhattan and theoretical non-rectangular networks which have different configuration and are used in various industrial fields. These network topologies have to be specifically developed in simulators. If there more such custom network topologies are to be developed, more time is needed to find a proper one. Hence, to accelerate the process of searching proper networks, we propose a technique to enable a quick check of the energy and time performance of non-Manhattan and theoretical non-rectangular topologies for specific applications by emulating data transmission of those topologies on a regular mesh instead of developing the networks one by one. Our technique models non-Manhattan and non-rectangular topologies on a grid-based NoC simulator NIRGAM to reduce the design cycles of simulating these different topologies.

As per [25], [169] and [6], a topology with a small diameter, small node degree and large bisection width is generally preferred to offer high throughput, low latency and energy performance, which is ideal to most applications. Yet such an ideal topology is hard to implement since trade-offs between hardware design cost and performance always exist. For example, a higher network degree refers to more channel links in a network node that leads to higher network throughput. But the design complexity of the more complex router structure and the extra area required for the router is increased. For this reason, a two-dimensional mesh is one of the most common topologies used by many automation frameworks for performance simulations due to its modularity and straight-forward layout. Other topologies, especially application-specific ones, are often customised for given applications, resulting in more desirable performance of the networks under specific scenarios.

To reduce the long design cycle, current researchers create different networks at system level and evaluate their performance in simulators for fast selection. However, modern simulators only support the automation of several popular architectures (like Mesh, Crossbar and Torus), leaving many custom topologies to be developed specifically. Considering this, we are motivated in this chapter to alleviate this gap between demands and simulator supports by using modern design automation techniques. As a result, high-efficiency modelling for virtual topologies is developed based on a modular mesh network, offering a fast emulation of specific custom topologies and routings with precise timing and energy performance simulations. Such a fast emulation is achieved by allocating different packet sending intervals and waiting time spans at different Mesh nodes to emulate the performance of different network topologies. The topological emulation reduces the design cycles in selecting custom topologies for specific applications. To validate our approach, two non-rectangular topologies, a honeycomb hexagon and a sparse-octagon, and two irregular topologies are explored by using our method as the examples. Their performance is also compared with the real networks for functional validation.

4.1 Introduction and Motivation

Modelling appropriate NoC network topologies for different applications greatly impacts on system performance and design cost [3]. In modern design automation tools, two methods of NoC architectural design are commonly used. One is using homogeneous building blocks to construct regular network architectures, which facilitates module reuse to reduce design complexity and cycles. The other is designing ad-hoc topologies with diverse specifically-designed modules that results in highly custom and irregular architectures with better performance and longer design cycles than the previous method.

However, due to the increased performance demands and design complexity of various NoC topologies, a conflict between network design for performance and design for productivity has emerged in modern topological automation techniques. Automatic generation of generic rectangular topologies like Mesh and Torus [36] are supported by many mainstream NoC simulators due to the architectural simplicity and reusable module generation. But these topologies can no longer fully satisfy the performance demands of more complex applications. Instead, non-rectangular topologies (like hexagon, octagon and spidergon) and irregular geometries are increasingly needed and expected to offer desirable performance and functionality for specific designs. Hence, the increase in architectural diversity and performance demands of NoC topologies for more specific and complex applications is at the cost of the decrease in topological automation and design efficiency of specific modules.

Several modern design automation tools have proposed techniques to support the construction of specific network topologies. Chen *et al.*[170] advocated a heterogeneous design methodology that provides a trade off between regular and irregular topologies in homogeneous networks with 3 sub-topologies: Ring, Octagon and 2-hop. Neeb *et al.*[171] proposed a customised irregular network-on-chip, *INoC*, tailored to specific applications. Choudhary *et al.*[169] proposed a solution to generate irregular topologies by making distributed table-based routing algorithms adaptive to irregular networks. Yet this previous work offers uniquely designed modules with highly specific functionality for the automatic generation of certain non-rectangular and irregular topologies. These specific modules are separated from regular schemes, requiring extra time to the module design with low reusability. So these simulation tools provide design automation of more custom modules for meeting performance requirements of certain topologies, without considering much for the reusability and consumed design cycles of those unique modules. To our best knowledge, a compromise solution of modelling non-rectangular and other specific topologies with satisfied performance while considering their module reusability is still absent. To fill the gap, combining the merits of those two common methods while overcoming their shortages, we are motivated to develop a new method to provide topological automation designs with both reusable modules and precise performance.

To utilise our proposed automation technique in practice, as the design flow shown in Figure 1.1, Chapter 1, we could rapidly emulate custom topologies and evaluate their energy and timing performance on a specific application. By using a modular Mesh network, our method could attach time-regulated models onto different Mesh nodes to change the packet waiting time spans and sending time intervals of these nodes. The timing cost of data routing through these channels can thus be manually altered, which can emulate similar performance of the same data on different topologies. In a design flow of NoC architectures for specific applications like in Figure 1.1, our method can quickly emulate a number of candidate NoC architectures on a mesh network instead of developing each network and evaluating their performance step by step, which enables efficient performance analysis of those candidate networks at early design stages. Such efficient analysis and estimation may save considerable design cycles in finding the optimised NoC topology for a specific application.

It is noteworthy that our proposed method only emulates different NoC architectures virtually and generates fast simulations of energy and time for performance estimation, rather than developing those networks physically. Hence the generated virtual topologies cannot be used for the physical implementation phase. Once an optimised NoC architecture is decided, it still needs to be developed following common design process from RTL to physical implementation.

The remainder of this chapter is organised as follows: The significance of modelling non-rectangular NoC architectures is firstly briefed, followed by the system-level development of two example topologies, honeycomb hexagon and sparse-octagon, in NIRGAM in the custom way. Then the proposed method is introduced by virtually modelling those two example topologies with our time-regulated models in the simulator. The specific routing schemes and model configuration for these networks are also given. After that, specific applications like data routings in both virtual and real non-rectangular and irregular geometries are experimented with the comparison of their timing and energy performance. For functional validation of the two virtual example topologies, experiments under random and hotspot traffic scenarios with a number of injection rates are also implemented to compare the timing, throughput and energy performance with real networks. The advantages and limits of our time-regulated models and topological emulation method are thus explored based on those simulation results. Next, an MPEG-4 decoder application is applied for operating on mesh with irregular routings built by both our method and the custom way are explored the potential of our method for multimedia applications. Finally, the superiority of our emulation method for the high-level modelling of various NoC architectures is summarised. Some conclusions are drawn at the end.

4.2 Non-rectangular NoC Architectures

Before showing our method of time-regulated models for the efficient architectural emulation in NIRGAM, the significance of system-level design automation of non-rectangular NoC topologies is firstly explained in this section. Then the conventional way of customising these two non-rectangular NoC architectures in NIRGAM is developed for comparative experiments in subsequent sections.

4.2.1 Significance of Non-rectangular Topological Modelling

Specifically, NoC architectures of a honeycomb hexagon and a sparse-octagon are shown in Figure 4.1. The reasons why non-rectangular topologies are of strong interests are worth clarifying. The first reason is these two example topologies represent non-rectangular topologies with competitive potential compared to rectangular networks like mesh topology. As per [3] and [172], a number of characteristics are considered to be desirable for a network topology such as high throughput, low latency, low power consumption, low area and high implementation feasibility. These characteristics often mutually contradict. A trade off, called the network cost between network degree and network diameter, is common in NoC topology designs for measuring the implementation feasibility. Its equation is introduced in [3] and shown below.

$$NetworkCost = Degree \times Diameter \tag{4.1}$$

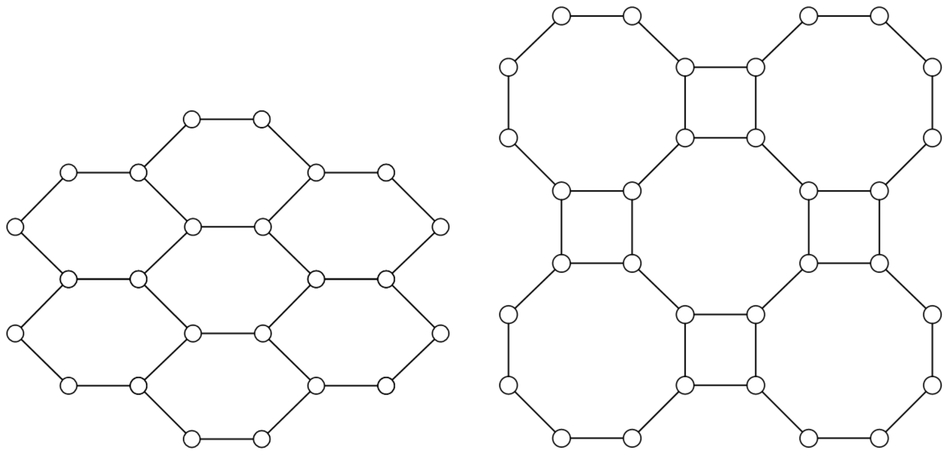

(a) Honeycomb Hexagon (b) Sparse-octagon

Figure 4.1: The Exemplified Honeycomb and Sparse-octagon Topologies

The network cost parameter multiplies degree, which refers to the number of channels each node has (that impacts on hardware cost), with diameter, which is the longest minimal node hops of network routings (that impacts on data transmission time) [173]. Non-rectangular topologies have good network cost compared to rectangular ones like Mesh. For the two example topologies, this parameter is even better than Mesh, as shown in Figure 4.2. In particular, honeycomb hexagon and sparse-octagon topologies have an average of 36.59% and 20.03% savings in network cost compared to the Mesh, suggesting the competitive potential for NoC designs.

Another reason why we have interests on these two topologies is that they can be transformed to rectangular shapes. As shown in Figure 4.3, such brick shapes of honeycomb and sparse-octagon topologies not only keep their architectural and characteristic advantages, but also offer good implementation feasibility. As mesh/rectangular based layout, implementation and placements are dominant in modern IC fabrication industries, applying brick-shaped non-rectangular topologies for implementation would be highly efficient usage of existing resources, compared to those special designs for specific custom NoC architectures.

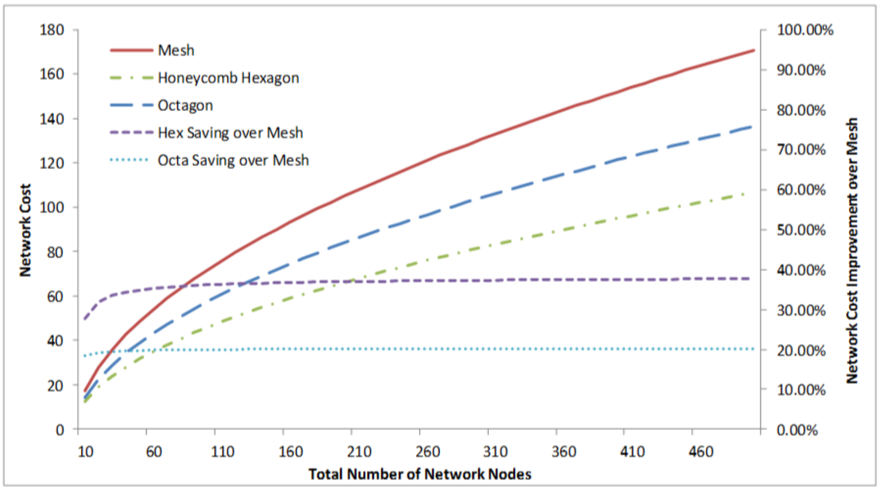
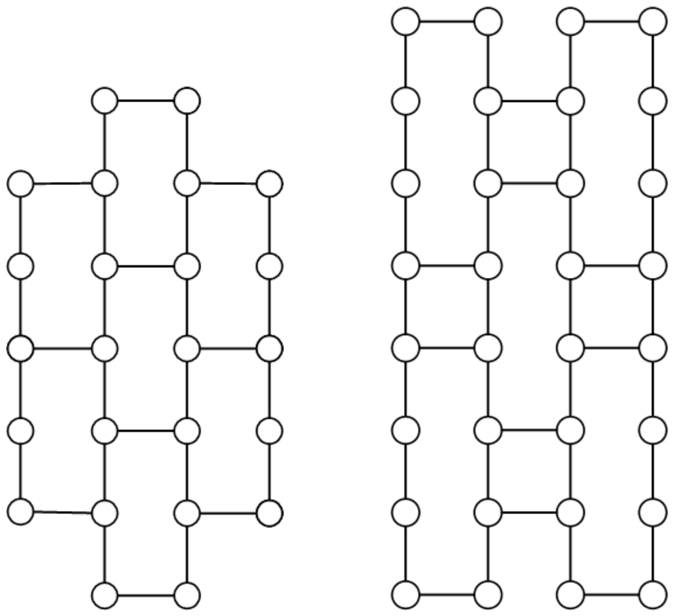


Figure 4.2: Network Cost Comparisons between Mesh and Non-rectangular Topologies



(a) Brick-Shape Honeycomb (b) Brick-Shape Octagon

Figure 4.3: Brick-Shape Transformations of Non-rectangular Topologies

For above reasons, although there are not yet too many applications for such type of NoC architectures, non-rectangular topologies are still competitive in achieving desirable network performance and hence worth implementing in system-level design automation. Moreover, since only a few modern simulation tools provide custom topological construction for the non-rectangular networks, long design cycles are required for designing those unique modules. It would be meaningful to use our method to efficiently emulate such networks and achieve their performance for massive applications.

Besides, the regular interconnection and competitive network characteristics make non-rectangular topologies easy to emulate by the reusable modules of a mesh network. However, it is worth pointing out that non-rectangular topologies only demonstrate part of the functionality and usage our model can provide. Custom irregular routings can also be virtually emulated by our models for certain applications and design requirements. Both regular and irregular routings are experimented with in subsequent sections to display the full functionality of our method.

4.2.2 Conventional Formation of Honeycomb Hexagonal Network

As a reference for later comparative experiments in Section 4.4, a real honeycomb hexagonal topology is modelled at system level via the conventional method. Specific modules and links from the mesh network built in the NIRGAM simulator are used as the reference network. Referring to a rectangular coordinate system, we re-design the homogeneous nodes of a default mesh network in NIRGAM in terms of their router structure, I/O ports and network interface module. Two types of hexagonal nodes that have 3 directional links instead of the default 4 links (North, South, West and East) to neighbouring nodes are developed, which are:

- Type 1 nodes: 60° North of East, 60° South of East, West;
- and Type 2 nodes: 60° North of West, 60° South of West and East.

Other unused mesh nodes are disconnected and removed from the honeycomb architecture. The channel capacity of all hexagonal node links are set to be equal to default mesh nodes in NIRGAM configuration files for the same network scale.

Figure 4.4 outlines a 24-node honeycomb topology constructed by the conventional method. Solid-line network nodes represent the used mesh nodes for the two types of hexagonal nodes. It is obvious that both types of hexagonal nodes have 4 channel links including 3 neighbouring directions (different in each type) and 1 IP core link, which is one link less than normal mesh nodes. The dash-line network nodes in the figure represent the unused nodes of the mesh network, which are thus disconnected and removed from the architecture. Since the arbitrary mechanism of hexagon is different for Mesh, designs of new routing algorithms are required to fully utilise this non-rectangular topology. Particularly in our case, an XY-like routing algorithm that is similar to traditional dimension order routings in the mesh network is developed based on previous work in [174] for deadlock-free routings on the honeycomb hexagon.

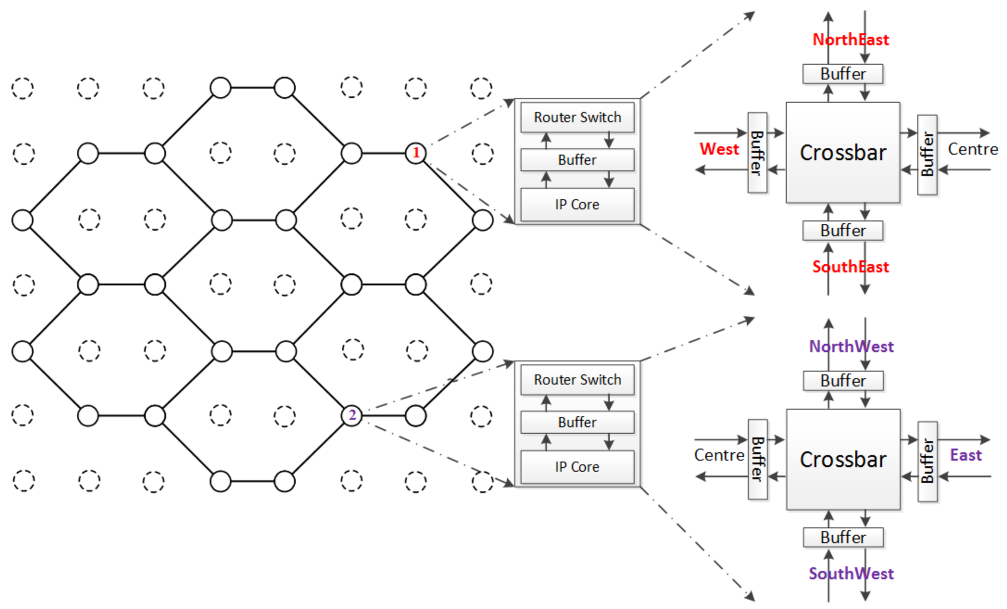


Figure 4.4: Conventional Formation of Honeycomb Topology

4.2.3 Conventional Formation of Sparse-Octagonal Network

To validate the functionality of our emulation method, a physical sparse-octagon network is constructed in NIRGAM by using modules with the same parameter settings as default mesh nodes. This architecture has a lower network cost than Mesh. Compared to the initial octagon architecture given in [175], our proposed octagon has no crossing links inside the loop to save area and to lower the design complexity. The topology is thus *sparser* than for a normal octagon and so is called *sparse-octagon*. Unlike hexagonal nodes, the octagonal nodes have 2 out of their 3 neighbouring links equal to the links mesh nodes possessed, leaving only one neighbouring link differing from normal mesh neighbouring links. The specific 4 types of octagonal nodes are:

- type 1 nodes have East, South, North West and Centre directions;
- type 2 nodes have West, South, North East and Centre directions;
- type 3 nodes have East, North, South West and Centre directions;
- type 4 nodes have West, North, South East and Centre directions.

The four types of octagonal nodes including their specific router structures, network interfaces and I/O ports are demonstrated in Figure 4.5. As observed, a 32-node sparse-octagon topology is constructed using a referenced mesh network. Solid-line nodes represent the octagonal nodes with specially designed modules, links and structures. Dash-line nodes indicate the unused mesh network nodes that are completely disconnected and removed. An adaptive routing algorithm with integrated deadlock recovery mechanism is derived from [176] for our sparse-octagon topology.

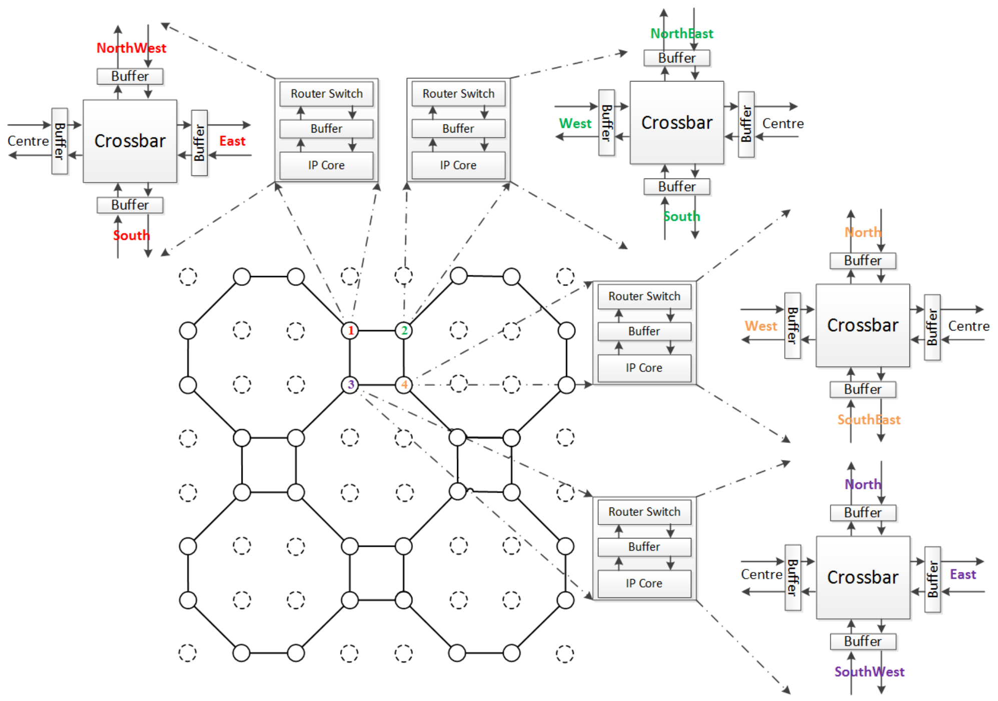


Figure 4.5: Conventional Formation of Sparse-Octagon Topology

4.3 Proposed Design Methodology

In this section, a *time-regulated model* is designed and seamlessly attached to the homogeneous mesh nodes in the NIRGAM simulator. The model can regulate the data sending time and injection rate at the output ports of attached nodes. A buffer is included in the model and its read pointer can be flexibly called to configure the output clocks, making it asynchronous to the input clocks. Thus, the data processing time span at nodes can be manually adjusted to accurately regulate the transmission times between adjacent node pairs. If the global transmitting time is regulated in a specific manner, data routing will be emulated as specific topological shapes, which then virtually forms the expected network topologies.

The two non-rectangular NoC architectures, honeycomb hexagon and sparse-octagon, are virtually emulated using our time-regulated model in this section. Moreover, functional validation of the virtual non-rectangular topologies is important to estimate the usability of the proposed method. In particular, energy and timing performance estimates of the emulated topologies are considered in our case. Hence, we have applied the modified Orion models integrated in NIRGAM for energy estimates. Subsequently, the principle of using a time-regulated model for the topological emulation and architectural modelling is explained in the honeycomb hexagon and sparse-octagon examples.

4.3.1 Proposed Time Regulated Model

The first step of the proposed method is to attach time-regulated models to specific nodes for precisely regulating sent data clocks. The values of sent time stamps can be adjusted by users in configuration files and all the different sending timers are derived from a global system clock to avoid clock jitter.

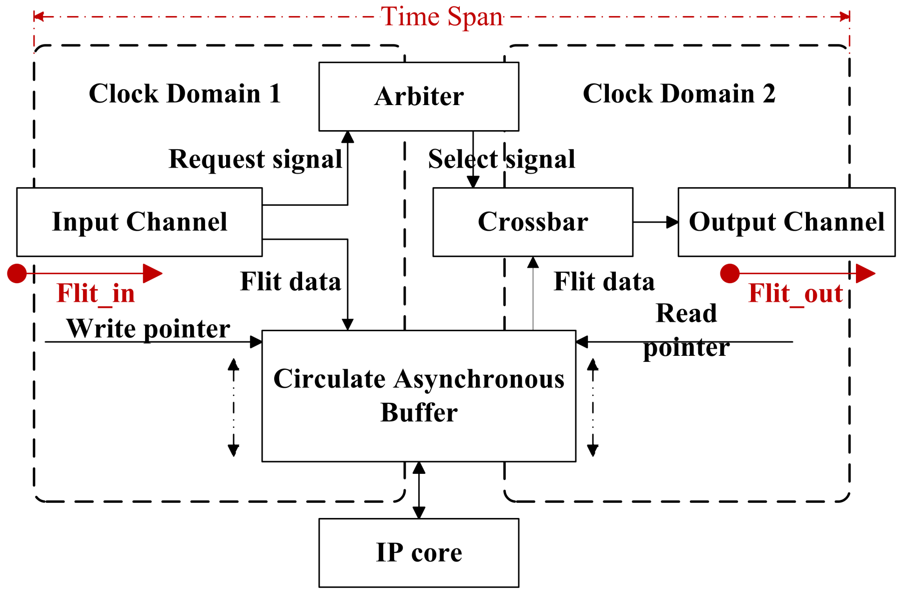


Figure 4.6: The Node Architecture with Time-regulated Model Attached

Figure 4.6 shows the detailed architecture of the time-regulated model. When a flit arrives at the input channels of heterogeneous nodes (mesh nodes whose time configuration is adjusted by the attached time-regulated model), a request signal is firstly sent to the router to ask for arbitration of the next-hop direction. The arbiter decides the direction depending on the destination identifier information contained in the flit and initiates a select signal to the crossbar for the directional indication. Flit data stored in the asynchronous circular buffer takes up 3 addresses for 1 flit indicated by buffer pointers, which are the packet identifier (pkt ID), flit identifier (flit ID) and flit content for each buffer address, respectively.

This whole process, except the read pointer, is controlled by the receiving clock, shown as clock domain 1 in Figure 4.6, which is also the same sending clock of the previous node. The buffer read pointer and select signal sent from the crossbar to output channels both operate under an asynchronous clock shown as clock domain 2. Clock 2 can be defined and modified in configuration files.

Once an asynchronous clock 2 is given, the time span between a flit travelling in and out of node is adjustable. In homogeneous nodes (mesh nodes whose time configuration is unchanged), the time span depends on the flit injection rate since the buffer read pointer needs to wait until the next clock cycle for sending a flit to the output channel, while in heterogeneous nodes the asynchronous buffer can reset the clock cycle immediately and initiate a sending process. If a specific time delay is inserted in front of the sending process, a larger or smaller time span will be represented as a time delay or time advance in the time line of flit traverse.

One step in establishing a non-rectangular topology based on a regular mesh network is attaching such functional models to network nodes and precisely regulate the time stamps for injecting sent data from specific node output ports. In such a heterogeneous node, the input clock cycle is equivalent to the sending clock cycle of the previous node, while its sending clock cycle is asynchronous. In other words, the received time and sent time at all node ports have the same length as a clock cycle, but adjusted to the different time stamps at which data starts to send from node outputs. The modified values can be adjusted in configuration files for accurate change of time spans at routers. Additionally, it is worth pointing out that all the node sending timers are derived from a global system clock to avoid clock jitter.

The values of the time delay or advance plus configurable sending cycles will then help regulate the flit traversal time between a node pair. In subsequent sections, examples of emulating two different non-rectangular topologies, honeycomb hexagon and sparse-octagon, by using this time-regulated model are given.

4.3.2 Example Topological Formation: Honeycomb Hexagon

In this subsection, the modelling of a virtual honeycomb hexagonal topology [177] using the proposed models is introduced as an example. In NoC data routings as per [3], several performance metrics are considered to measure the network between any node pair:

$$Throughput,\ Latency\ and\ Power(Energy)$$

where *Throughput* indicates the number of flit bits successfully traversing between node pairs over a particular time duration; *Latency* indicates the time consumed between first bit of a flit leaving one node and the last bit of the flit leaving the second. *Power* indicates the power consumption of all the bits of a flit traversing in the network.

Throughput is often controlled by the available bandwidth of the network and hardware limits. Once the network channel bandwidth is fixed, the maximally possible number of bits traversing within one second is constant along channel links between node pairs. The latency of flit data bits between a node pair may consist of multiple parts in packet-switched NoC networks. One part is the speed or velocity of a flit data bit which is limited by the speed of light, giving a minimum propagation delay in all medium that cannot be reduced any more. The transmission time on channel links between node pairs are often caused by such a propagation delay. Other parts of the delay commonly occur at the intermediate nodes of a routing path, including queueing delays at port buffers for directional arbitration, clock triggering and port usage. Such delays often occur at node routers, forming another significant part of latency that can be precisely adjusted by our time-regulated model.

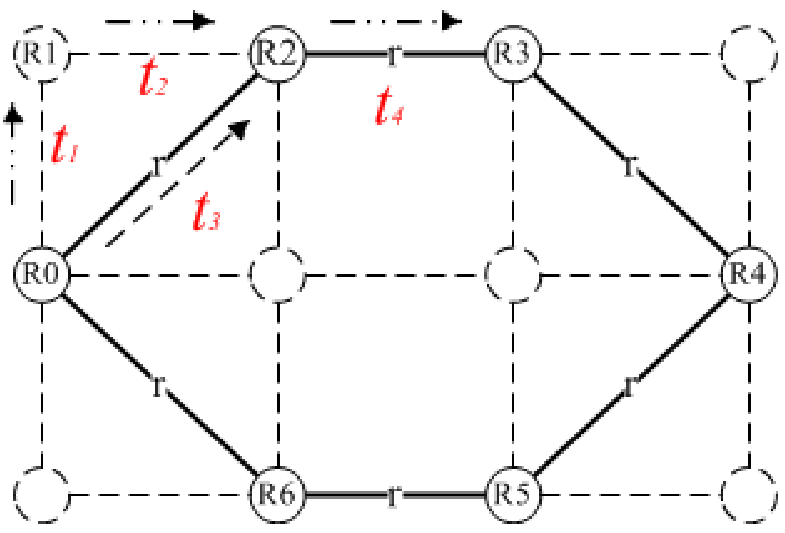
Since a mesh network contains the same switching mechanism in both homogeneous and heterogeneous nodes, the channel capacity (link bandwidth) of the whole network is fixed, indicating the same maximal amounts of flits (throughput) routed over the whole network. Hence, by attaching time-regulated models onto network nodes and quantatively setting the transmission time delayed at intermediate routers, our method could adjust (reduce or increase) the latency of data routings in a network without affecting its throughput performance.

Figure 4.7 shows the detailed process of modelling a virtual hexagonal topology on a mesh network using the proposed method. Figure 4.7(a) shows how the hexagonal shapes are virtually emulated. Assume each hexagonal side has an equal length of r units (for data traversal). Taking the hexagonal sides $R0 - R2$ and $R2 - R3$, for example, flits travelling between $R0-R2$ are expected to cost the same time delay as between $R2-R3$. If we denote the flit traversal time duration between node pairs $R0-R1$, $R1-R2$, $R2-R3$ and $R0 - R2$ in hexagon as t_1, t_2, t_3 and t_4 respectively, the **expected** hexagonal flit traversal time relation is:

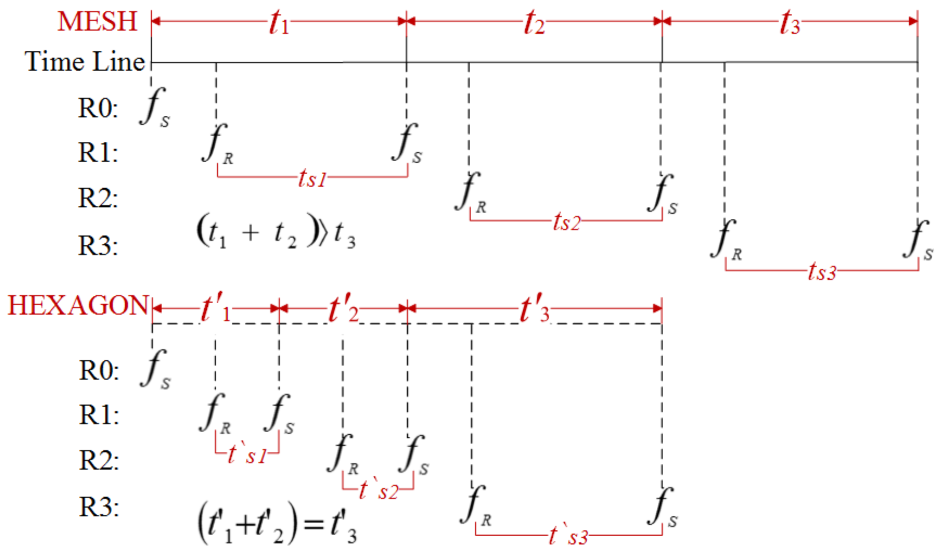
$$t_1 + t_2 = t_4 = t_3. \tag{4.2}$$

To implement this, time-regulated models are inserted at nodes $R0$, $R2$ and $R3$ of a mesh network. Accurate values of the time delay or advance are configured at each of the node buffers to reset send time stamps for equivalent flit traversal time spans between node pair $R0 - R2$ and $R2 - R3$. In other words, the transmission latency at node pair $R0 - R2$ is halved by reducing intermediate routing delays to make it equal to the transmission time cost at node pair $R2 - R3$, modelling such two routing paths to emulate a pair of hexagonal sides in a mesh topology.

Figure 4.7(b) gives the configuration process. In a regular mesh network, the flit traversal time between node pairs $R0-R1$ and $R1-R2$ is double that between $R2-R3$ ($t_1+t_2 = 2t_3 > t_3$), and their in-node processing time spans are equal ($t_{s1} + t_{s2} = t_{s3}$). To adjust the processing time, a smaller time delay is set to shorten the time span at node $R1$ ($t'_{s1} < t_{s1}$), which causes a send time advance and reduces the value of t_1 in the time line ($t'_1 < t_1$). Moreover, another smaller time delay is set to reduce the time span at node $R2$ ($t'_{s2} < t_{s2}$), which causes a send time advance and reduce the value of t_2 ($t'_2 < t_2$). The latency between nodes $R2-R3$ remains unchanged. Thus, the latency at respective node pairs $R0 - R2$ and $R2 - R3$ has been equalised ($t'_1 + t'_2 = t'_3$) that costs two-hop transmission time as in a hexagon while a normal mesh will cost three-hop time delays for data traversing from node $R0$ to $R3$. Similarly, values of flit traversal time in node pairs $R3 - R4$, $R4 - R5$, $R5 - R6$ and $R6 - R0$ are also adjusted to be equivalent.



(a) Mesh-based Hexagonal Topology



(b) Flowchart of Flit Traverse Task

Figure 4.7: Construction of Mesh-based Hexagonal Topology

The respective flit traversal latency in those node pairs are mutually equal, which emulates a directional loop in a virtual hexagonal topology. To make bidirectional loops both emulate hexagonal topology, proper routing schemes are designed to cooperate with relevant routing algorithms. Specifically for the honeycomb hexagon modelled by our method, two types of network nodes are identified, as shown in Figure 4.7(a), with hexagonal nodes (solid-line nodes) and mesh nodes (hash-line nodes). The special routing scheme on these two types of nodes are developed in order to suitably use deterministic mesh routing algorithms for the virtual hexagon topology, which is shown below:

- For time-regulated models attached to virtual hexagonal nodes,
  - the output time configuration is set to be equal to normal mesh nodes if the next hop of the data traversal is also a virtual hexagonal node;
  - the output time configuration is set to reduce the time span to half the routing delay of mesh nodes if the next hop is a mesh node.
- For time-regulated models attached to normal mesh nodes,
  - the next hop direction cannot be in the same direction as its current hop direction;
  - the next hop must be a virtual hexagonal node;
  - the output time configuration is always set to reduce the time span to half the routing delay of normal mesh nodes.

It is noteworthy that without developing specific router structures and functional modules for each different topology, a specially designed routing scheme combined with existing mesh routing algorithms is required by our method for correctly configuring the time-regulated models and thus properly routing data on the virtually modelled networks. This is the only extra element that is needed by our method compared to the custom way of constructing specific topologies, as described in the last row of Table 4.4.

Using the same principle, many other non-rectangular and irregular routing patterns can also be virtually emulated. Furthermore, if we employ an arbitrary number of heterogeneous node pairs, a virtual pseudo-irregular network topology could be achieved. If set the latency between heterogeneous node pairs and physical intervals between homogeneous node pairs to conform to regular patterns, some well known non-rectangular networks like SPIN [178], Spidergon [179], Hexagon [180], Octagon [175] [181] and Fat Tree [9] can be virtually formed.

4.3.3 Example Topological Formation: Sparse-Octagon

Another example of non-rectangular topologies, Sparse-Octagon, is also virtually emulated its topology on a mesh network by using our modelling method. As shown in Figure 4.8, time-regulated models are inserted to normal mesh nodes. Similarly to the virtual hexagonal emulation, the eight virtual sides of the octagonal network could be divided into 4S identical groups for regulating the equal traversal time of flits. Take node pairs $R0 - R1$ and $R1 - R2$ for example, the realistic flit traversal path goes from $R0$ to $R1$ then reaches $R2$ and $R3$ in order.

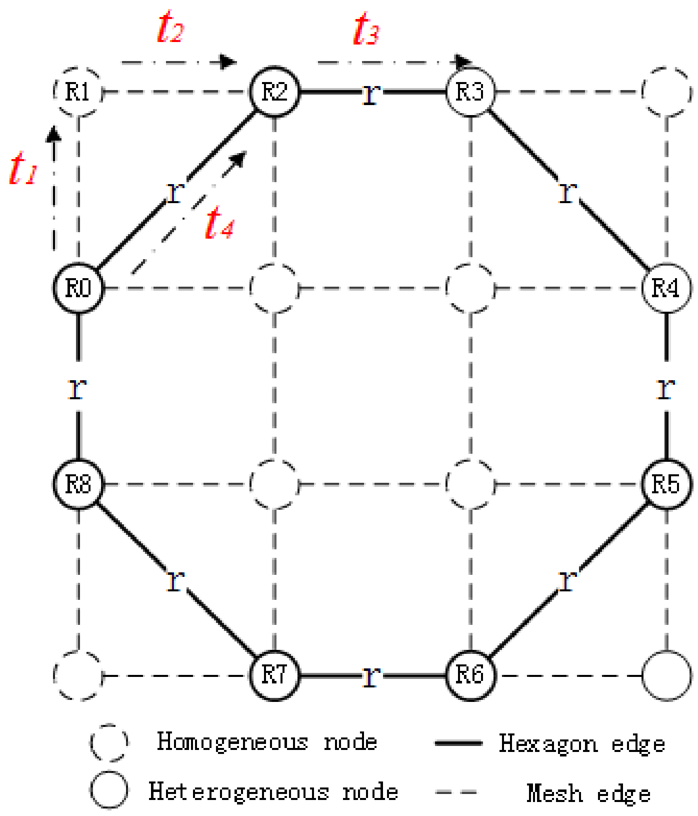


Figure 4.8: Mesh-based Octagon Topology

Giving the reduced time spans at node $R0$ and $R1$, the traversal time cost between node pairs $R0 - R1$ and $R1 - R2$ is configured to be equal to node pair $R2 - R3$ ($t_1 + t_2 = t_3$). Thus the latency between node pairs $R0 - R2$ equals that of node pair $R2 - R3$ ($t_4 = t_1 + t_2 = t_3$), emulating a virtual side of one-quarter basic unit of the octagonal topology. The other three groups of node pairs can configure their router time spans in similar ways to model the unidirectional octagon network. It is worth pointing out that the area occupied by one basic virtual octagonal unit (16 nodes, 4×4 mesh network) is larger than that of a virtual hexagonal unit (12 nodes, 4×3 mesh network), which implies a potentially smaller scale of topological emulations by using the proposed method for some large-size applications. A potential solution to fold the this specific non-rectangular topology may be to emulate the brick-shape transformation of the network. A generalised method to control the emulation area cost of the mesh network for various NoC architectures is still absent and required future research.

Similar to the modelling of the virtual honeycomb topology, bidirectional loops in the virtual octagonal architecture also need cooperation between the specifically designed routing scheme and modified mesh routing algorithms to function correctly. But the octagonal topology has 4 types of nodes with different neighbouring links instead of the 2 in honeycomb. As shown in Figure 4.8, solid-line and dash-line nodes again represent octagonal and mesh nodes respectively. Each octagonal node contains 4 total direction links including 3 neighbouring links and one IP core link as well. The special routing scheme for virtual octagon is similar to honeycomb, which is:

- For time-regulated models attached to all types of virtual octagonal nodes,
  - the output time configuration is set to be equal to normal mesh nodes if the next hop of data traversal is also a virtual octagonal node;
  - the output time configuration is set to reduce the time span to half the routing delay of mesh nodes if next hop of data traversal is a mesh node.
- For time-regulated models attached to normal mesh nodes,
  - the next hop direction cannot be in the same direction as its current hop direction;
  - the next hop must be a virtual octagonal node;
  - the output time configuration is always set to reduce the time span to half the routing delay of normal mesh nodes.

Although the routing schemes of octagon and hexagon are similar, it should be pointed out that configuring time spans of intermediate mesh nodes to half of normal mesh nodes is only proper for such types of topological emulations. It is because in both honeycomb and sparse-octagon architectural emulations we model two mesh hops to cost equal time delays as one hop along non-rectangular network sides. Other topologies especially irregular routing patterns may require different value settings of time span configuration, resulting in shorter or longer delay performance rather than half of the normal mesh routing delay. The quantative adjustment is determined via the careful learning of network characteristics, which is in correspondence with the first row of requirements inTable 4.4.

4.4 Experimental Results

The functionality of our proposed time-regulated model needs to be validated under different scenarios. Firstly, since non-rectangular NoC topologies of honeycomb and sparse-octagon have been virtually emulated by our method as examples, they are used for the validation. Moreover, irregular routings are also designed and emulated by our method to test the functionality. Specifically, given a pair of source and destination with a fixed Euclidean distance, both exemplified non-rectangular routings (honeycomb and octagon) and the two irregular routings are deployed to connect the source and destination nodes in different topological geometries. The transmission time between the source and destination nodes are both theoretically calculated and experimentally implemented to inspect their result consistency. As a baseline, these virtual routings (non-rectangular and irregular) are compared with a mesh rectangular routing (deployed on the same source-destination pair). Besides, the network performance of those virtual topologies and geometries in terms of latency and energy are also compared with the real networks constructed in the custom way. In a word, a fixed source-destination pair under the Mesh, two non-rectangular and two irregular geometries are constructed in a custom way and by our emulation method. Their routing performance are mutually compared to validate the results of virtual routings emulated by our method.

Two commonly used traffic patterns, uniform random and hotspot, are designed for the two exemplified virtual non-rectangular topologies to test if their network characteristics (energy and time) are consistent with the real ones. Moreover, based on the experimental results, the performance accuracy and limits of our proposed time-regulated model are summarised. Then, an MPEG-4 decoding application is implemented as a test case to explore the potential of our virtual topological modelling in real-world multimedia applications. The MPEG application is split into several IP tasks that are mapped onto a real irregular network in the custom way and onto a Mesh network with an virtual irregular routing emulated by our method. The energy and timing simulation results of the real irregular routing and the virtual irregular routing are compared. Finally, the superiority of our virtual emulation method over the conventional custom method is discussed based on all the case experiments.

4.4.1 Specific Routings: Hexagonal and Irregular Routings

4.4.1.1 Experimental Setup

Since the proposed method aims to provide emulations of non-rectangular and irregular geometries based on the mesh network, the potential performance difference of a fixed data routing simulated on those geometries is experimented and compared to the performance in a Manhattan geometry (i.e. Mesh). Assume a pair of nodes with the same Euclidean distance between the source and destination co-ordinates, data routings between the node pair along hexagonal, irregular and mesh geometries are given to validate the performance difference in terms of timing and energy. Theoretical distances of different topological routings between the fixed-length node pair are firstly calculated. Then these virtual routings emulated by our method are simulated in NIRGAM. If the simulation performance is consistent with theoretical results, the functional accuracy and applicability of our method can be verified.

The environment for subsequent experiments is implemented on a 9 by 9 mesh network. 4 virtual channels are created in each physical channel in all experiments to avoid performance degradation caused by head-of-line blocking problems. In all experiments, initial packet injection rate is given at one packet every 20 clock cycles, that is, the initial flit interval at the source node is kept as 20 clock cycles. The intermediate injection rate in all other nodes are all set at one flit every 100 clock cycles to avoid potential transmission interrupts and packet drops (In this case the injection rate is fixed for hexagonal routing, so there are no packet drops. But for other circumstances especially irregular routings, different intermediate nodes may have different injection rates which may incur packet drops if the injection rate at one node is too high to that data saturates the node buffer before sending). For each experiment, a different amount of data packets is generated at the source at the initial packet injection rate and sent to the destination node via different routing geometries at the intermediate injection rate. The amount of data packets ranges from 120 to 200 with 1-flit per packet for each different experiment. It is worth pointing out that the time spans cost at different network nodes in one network for each directional routing can be independently regulated to different values depending on specific design requirements. Thus it is easy to emulate transmission times at each node pair of an irregular routing path. As all the routings are of certain geometries, source routing algorithm is used to given certain routing path for experiments.

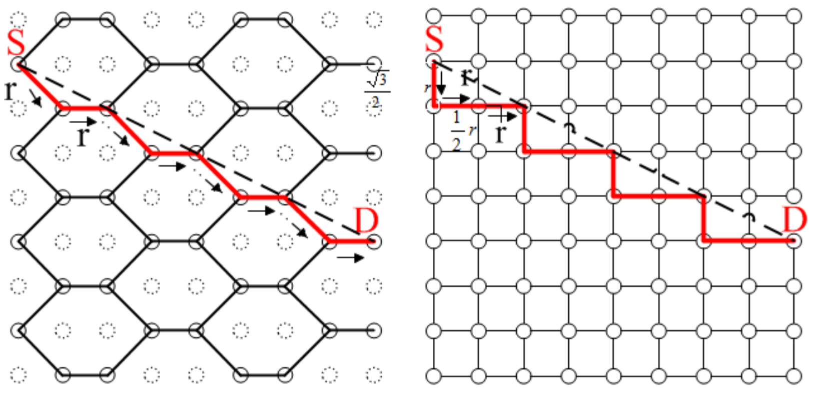
4.4.1.2 Theoretical Calculation

Data routings between a fixed-length node pair along the hexagonal, two irregular and the mesh network geometries are developed and shown in Figure 4.9 and Figure 4.10. These non-rectangular or irregular routings are implemented on both virtual and real topologies for performance comparisons. Figure 4.9(a) and Figure 4.9(b) show the hexagonal and mesh routings on virtual topologies emulated by the proposed method as well as on real topologies constructed in the custom way. In a virtual hexagonal network, the side length (radius) is set equal to r units. Based on the backbone mesh network, the coordinates of the source and destination nodes are set to $[0, 1]$ and $[8, 5]$ respectively. Thus, the theoretical hexagonal routing distance from the source to destination node can be derived as:

$$S - D_{[hexagon]} = 4r + 4r = 8r \quad units \tag{4.3}$$

Considering the same coordinate system in an orthogonal mesh network, flits need to travel $\frac{\sqrt{3}}{2}r$ units vertically and $\frac{1}{2}r$ units horizontally to reach the first Euclidean point in a hexagon. Then they need to traverse another r units horizontally to reach the second relative Euclidean point. The flits will arrive at the destination node after the above process repeats 4 times. Consequently, the theoretical Manhattan distance from the source node to destination node is:

$$\begin{aligned} S - D_{[manhattan]} &= 4(\frac{\sqrt{3}}{2}r + \frac{1}{2}r + r) \\ &\approx 9.5r \quad units \end{aligned} \tag{4.4}$$



(a) Data Routings in Virtual Hexagon Network (b) Data Routings in 2D-Mesh Network

Figure 4.9: Experimental Designs for Data Routing in Hexagonal and Mesh Networks

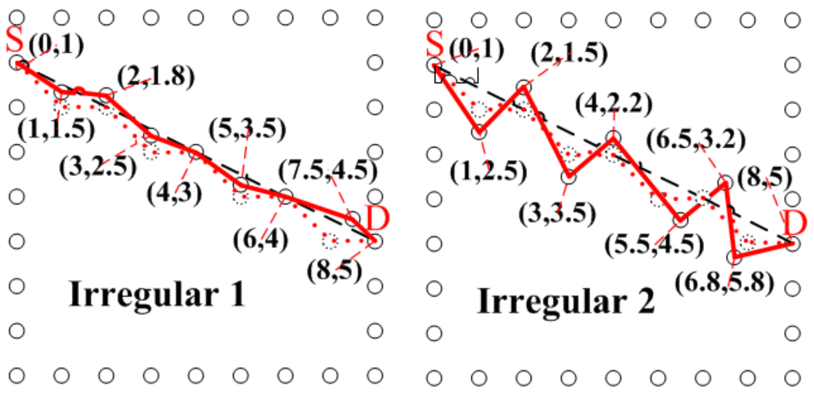
It can be seen that the data routing distance in a mesh is longer than in a hexagon. Based on the same coordinate system, the unique Euclidean distance, i.e. the straight line between the source and destination, which is also the shortest routing path is:

$$\begin{aligned} S - D_{[euclidean]} &= 4 \times \sqrt{(\frac{\sqrt{3}}{2})^2 + (\frac{1}{2}r + r)^2} \\ &\approx 6.9r \quad units \end{aligned} \tag{4.5}$$

Additionally, Figure 4.10(a) and Figure 4.10(b) show two irregular routings using the same nodes as in the hexagon and mesh. The node ID and data routing hops are intentionally set equal to the hexagonal routing for comparing the proportional differences of experimental performance with their theoretical calculations. The node coordinates and data routing time to their neighbour nodes are configured by the time-regulated model as shown in the figure. In both irregular networks, black dash lines, red dot lines and red solid lines indicate the Euclidean distance between the source and destination node, hexagonal routing path and virtual irregular routing paths, respectively.

It is observed that the irregular 1 path routes data more closely to the Euclidean straight line than the hexagonal geometry, while the irregular 2 path is less close, which suggests a shorter traversal distance for irregular 1 and a longer traversal distance for irregular 2 than the hexagonal distance. Based on the mesh coordinate system, their calculated traversal distances are:

$$\begin{aligned} S - D_{[irregular1]} &= \frac{3\sqrt{7}}{4}r + \frac{3\sqrt{19}}{4}r + \frac{\sqrt{427}}{20}r + \frac{\sqrt{547}}{20}r \\ &\approx 7.5r \quad units \end{aligned} \tag{4.6}$$



(a) Data Routings in Irregular1 Network (b) Data Routings in Irregular2 Network

Figure 4.10: Experimental Designs for Data Routing in Irregular Networks

$$
\begin{aligned}
S - D_{[irregular2]} &= \frac{\sqrt{31}}{4}r + \frac{\sqrt{7}}{4}r + \frac{\sqrt{13}}{4}r + \frac{\sqrt{907}}{20}r \\
&+ \frac{\sqrt{1987}}{20}r + \frac{\sqrt{732}}{20}r + \frac{\sqrt{2021}}{20}r + 1.3r \\
&\approx 11.6r \quad units
\end{aligned}
\tag{4.7}
$$

As per the calculated distances, flit routing in the virtual irregular 1 network is expected to have the least traversal time cost, followed by the virtual hexagon, mesh and virtual irregular 2 routings sequentially. To validate the emulated accuracy of the proposed method, all the virtual routings are compared with the same routings on real topologies constructed by the custom way in terms of timing and energy performance. Clearly, data routing in a mesh network should consume more energy and larger latency than in the hexagonal network due to its longer Manhattan routing distance. Given a fixed numbers of data messages, the difference in performance will be observed in terms of total communication energy and traversal time cost from the source to destination in both hexagonal and mesh networks.

4.4.1.3 Result Analysis

Figure 4.11 summarises the timing comparison of data routings in the virtual hexagon, mesh and the two virtual irregular geometries. The cycle time is 1 ns. The X-axis gives the total packets injected at each experiment and transmitted in the networks under different routing geometries. The Y-axis shows different performance metrics measured for the transmission of all those packets from the source to the destination node. The packets ranging from 120 to 200 are given for the result comparisons of different routings. As the results show, the time cost of data in all four proposed topologies increases linearly with the increase of transmission packets. The irregular 1 routing has the least time cost in all flit injection situations, sequentially followed by hexagonal routing, mesh routing and irregular 2 routing. The timing differences reflect their theoretical transmission time compared with the above distance calculations.

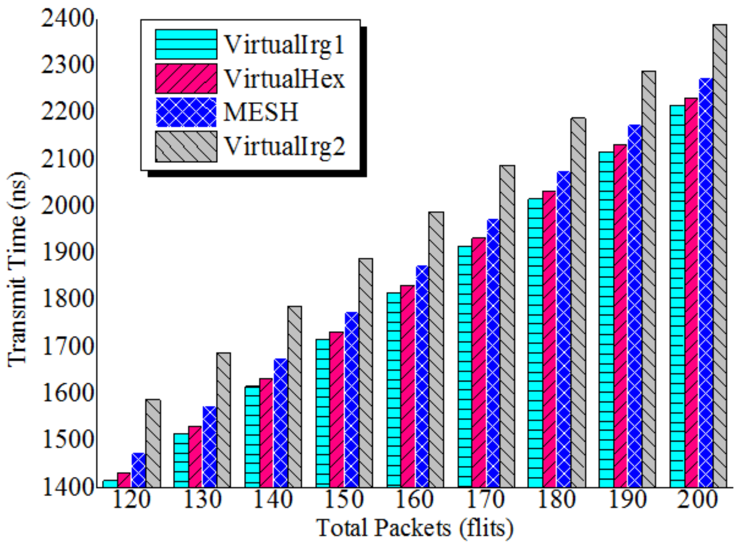


Figure 4.11: Performance Comparison of Hexagon Topology and 2D-Mesh in terms of Overall Transmission Latency based on 9 by 9 Mesh.

Figure 4.12 shows the energy results of four virtual routings over all flit injection cases. These energy results show a slight difference from the timing performance. The virtual irregular 1 routing consumes the smallest energy followed by virtual hexagonal, virtual irregular 2 and mesh routings, respectively. But the virtual irregular 2 routing consumes less energy than mesh routing which is inconsistent with the timing performance. The extra energy consumption of mesh routing is due to its more average flit hops than the other three.

As mentioned before, the two virtual irregular routings are intentionally constructed with the same nodes used for hexagonal routing, giving the same flit hops that costs the same dynamic energy for all three geometries. The energy differences between virtual irregular and hexagonal routings are mainly determined by their static energy differences which are proportional to their routing time differences. Consequently, although the irregular 2 routing consumes more static energy than the mesh routing due to its longer routing time, the extra dynamic energy cost by mesh routing still offsets and overwhelms their static energy difference.

The above experiments verify that virtual non-rectangular and irregular routings emulated by the proposed method can present consistent timing differences as their theoretical distance differences suggest. The following experiments are designed to validate the performance accuracy of virtual routings. The same data routings are compared on virtual topologies emulated by the proposed method and real networks

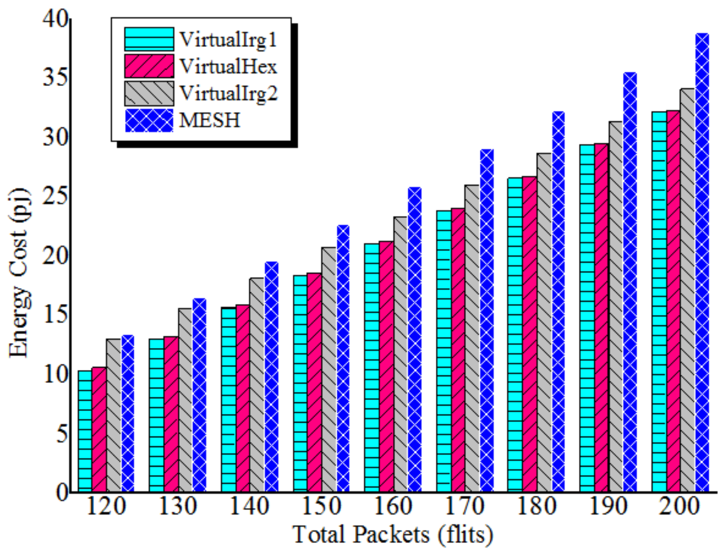
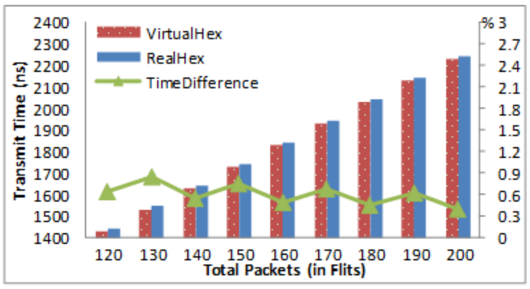
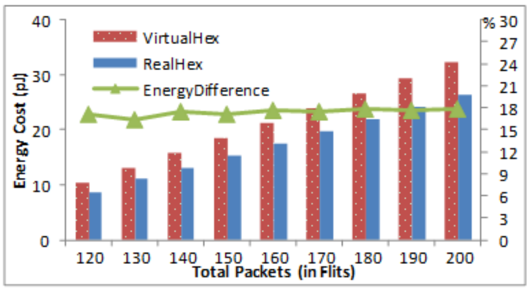


Figure 4.12: Performance Comparison of Hexagon Topology and 2D-Mesh in terms of Energy Consumption based on 9 by 9 Mesh

constructed in the custom way. Figure 4.13, Figure 4.14 and Figure 4.15 compare the timing and energy performance of routing data in a virtual hexagonal and two irregular geometries with in the real counterparts, respectively. As shown in Figure 4.13(a), Figure 4.14(a) and Figure 4.15(a), the timing performance of the virtual hexagonal and two irregular routings are, respectively, 0.7%, 0.8% and 0.3% different compared to the real hexagonal and irregular topologies, which indicates the virtual routings emulated by the proposed method could provide an accurate timing performance.



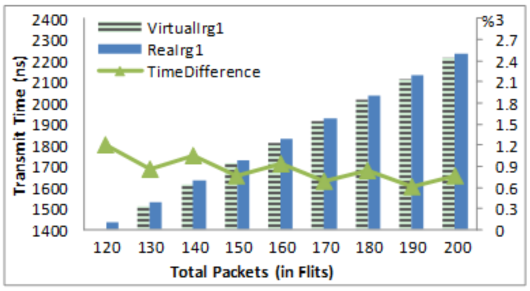
(a) Virtual and Real Hexagon Timing Comparison



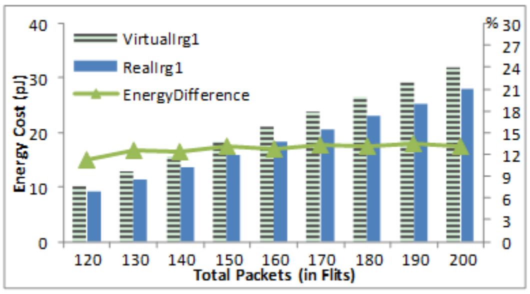
(b) Virtual and Real Hexagon Energy Comparison

Figure 4.13: Timing and Energy Comparisons of Data Routings in Virtual and Real Hexagonal Networks

As shown in Figure 4.13(b), Figure 4.14(b) and Figure 4.15(b), the energy performance of the virtual hexagonal and two irregular routings are around 17%, 12% and 0.2% different compared to the real hexagonal and irregular routings, respectively. The considerable inaccuracy of energy performance in the hexagonal and irregular 1 routings is due to

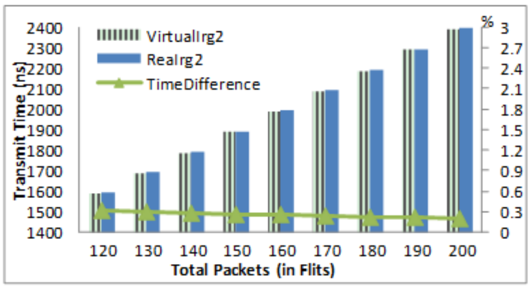


(a) Virtual and Real Irregular 1 Timing Comparison

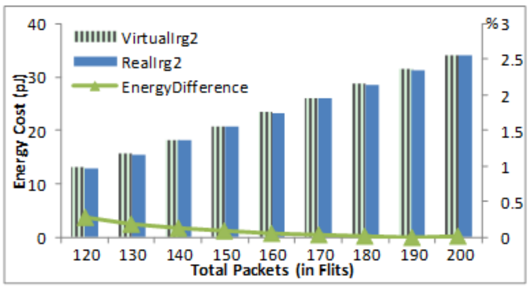


(b) Virtual and Real Irregular 1 Energy Comparison

Figure 4.14: Timing and Energy Comparisons of Data Routings in Virtual and Real Irregular 1 Networks



(a) Virtual and Real Irregular 2 Timing Comparison



(b) Virtual and Real Irregular 2 Energy Comparison

Figure 4.15: Timing and Energy Comparisons of Data Routings in Virtual and Real Irregular 2 Networks

the different flit hops consumed in the virtual and real routings; that is, the proposed method constructs these two virtual routings with more flit hops than the flit hops in their real counterparts. The extra hops bring out more flit switching at nodes, which costs more dynamic energy.

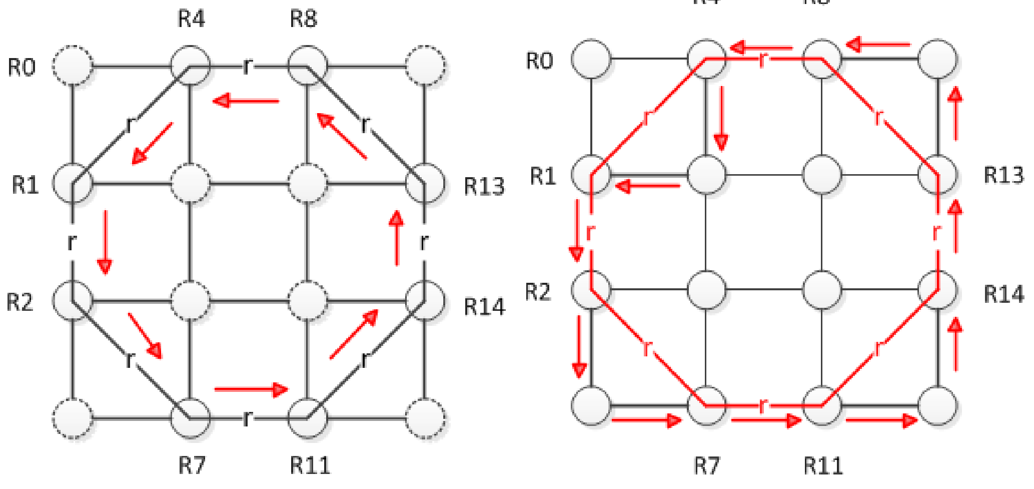
The irregular 1 routing comparison has less energy difference (12%) than the hexagonal routing comparison has (17%) since the extra hops used in the virtual irregular 1 routing are fewer than those used in the virtual hexagon. On the other hand, the hops of the virtual irregular 2 routing can be manually adjusted to be the same as in the real model, which causes the energy performance of the virtual irregular 2 routing to be very close (about 0.2% difference) to the real routing. The experimental energy performance indicates the virtual routings emulated by the proposed method can provide moderate energy performance accuracy. The accuracy could be considerably improved if the flit hops used by the virtual routings were close enough to the flit hops used in the real counterparts.

To summarise, the above comparative experiments first verified that the timing and energy differences among virtual hexagonal and irregular routings are consistent with their theoretical performance differences. Then the direct timing comparisons of routings in virtual and real geometries validated that the proposed method can provide accurate timing emulations to non-rectangular and irregular geometries in the given experimental scenarios. Finally, the energy comparisons between virtual and real routings suggest the accurate energy emulation is achievable by our emulation method if the hops of virtual routings can be emulated to approximate to the hops cost in real networks.

4.4.2 Specific Routings: Octagonal Routing

4.4.2.1 Experimental Setup

Besides hexagonal comparison experiments, a simple comparative experiment is also implemented to show the proper use of our proposed model for sparse-octagonal topology emulation. As shown in Figure 4.16, a 10-flit packet is transmitted circularly on an octagonal unit constructed in the custom way. The transmission starts from node $R4$ back to $R4$ again after a counter-clockwise transmission, while in a mesh network an equivalent traversal routing is implemented via our model to emulate the virtual octagon along the same circulation but with 4 extra hops. Besides, the flit circulation running on a normal mesh network that has the same numbers of hops as the virtual octagon is also used for the energy comparison. The total time and energy cost for the flit circulation in all three networks is recorded.



(a) Octagonal Routing (b) Mesh Routing

Figure 4.16: Experimental Designs for Comparing Octagonal and 2D-Mesh Networks

4.4.2.2 Result Analysis

The performance difference between the octagonal routing and the mesh routing is firstly inspected to validate the functionality of the octagonal network. Then the modelling accuracy of our virtual emulation method is examined by comparing the performance of virtual and real octagonal routings.

Theoretically, the same routing circulation implemented on a mesh and a sparse-octagon is expected to obtain different timing and energy costs. It is because if all other network characteristics are equal, the octagon will consume only 8 hops while the mesh needs 12 hops to finish the same routing. This hop difference impacts on both energy and timing performance, resulting in different performance between the two networks. Table 4.1 shows the performance comparison between octagonal and mesh routings in NIRGAM simulations. As shown, the real octagonal network costs 11.73% less time and 15.32% less energy than mesh, which is consistent to the theoretical inference.

	Energy Cost (pJ)	Time Cost (ns)
Real-Octagon	166.32	346
Mesh	196.41	392
Difference (R-O vs M)	15.32%	11.73%

Table 4.1: Comparisons of Octagonal and Mesh Routings

Table 4.2 lists simulation results of the virtual octagonal routing emulated on a mesh network and the real octagonal routing constructed by the custom way in terms of timing and energy. The performance errors of the virtual routing over the real one is also given. As seen, our virtual emulation consumes nearly the same routing time (merely 1.02% error), revealing precise and satisfied accuracy of our model for non-rectangular routings. The energy error is 14.96% due to the extra 4 hops consumed by virtual routing over the real one.

	Energy Cost (pJ)	Time Cost (ns)
Real-Octagon	166.32	346
Virtual-Octagon	195.575	342
Error (V-O vs R-O)	14.96%	1.02%

Table 4.2: Comparisons of Virtual and Real Octagonal Routings

So far the experimental results of these specific routings have displayed that both non-rectangular topologies and irregular routing geometries virtually modelled by the proposed emulation method could generates accurate energy and transmission time performance if the configuration of network parameters in terms of injection rate and routing hops are set properly. In next sections, the network characteristics of the two virtual non-rectangular topologies emulated by our method will be experimented with the comparison of real topologies in terms of time and energy.

4.4.3 Synthetic Traffic: Uniform Random

4.4.3.1 Experimental Setup

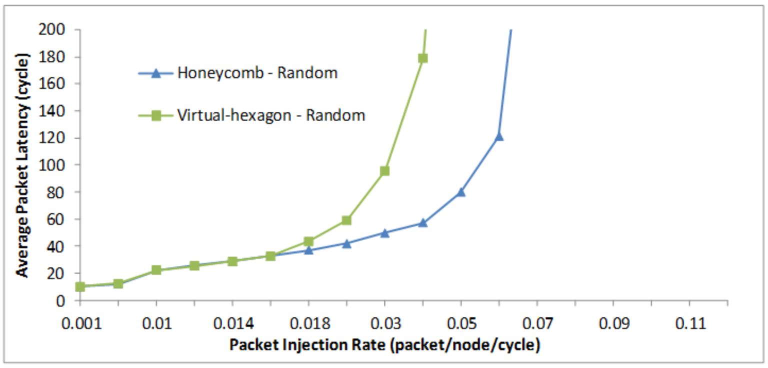
To validate our emulation method, the two non-rectangular NoC topologies modelled on a mesh in NIRGAM are compared with the real networks. An uniform random traffic scenario is firstly used. In this traffic, each node in the network generates a constant bit stream of certain packets and flits at its IP core, sending each packet to randomly chosen destination nodes with the same probability.

The network scale used for **honeycomb** hexagon has 24 nodes. A 24-node network constructed via the custom way is given for the real hexagon. A 7 by 8 mesh topology is built for emulating a virtual honeycomb by using our modelling method. As shown in Figure 4.4, the solid-line nodes are real hexagon network while all nodes (solid-line nodes plus dash-line nodes) represent the mesh network used for virtual hexagonal emulation. The network scale of **sparse-octagon** has 32 nodes. As shown in Figure 4.5, a 32-node real sparse-octagon is built on the solid-line nodes while an 8 by 8 mesh network is used (both solid-line and dash-line nodes) for modelling a virtual octagon topology. All the real and virtual non-rectangular networks are designed using the specific model configuration, routing schemes and routing algorithms, which were introduced above.

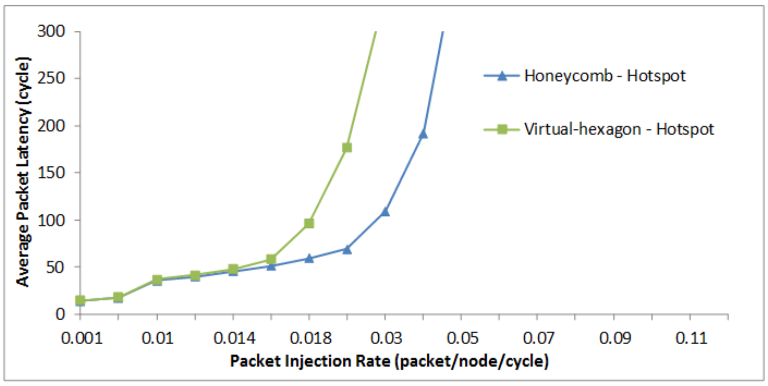
For performance evaluation, a range of packet injection rates (pir) from 0.001 to 0.12 with 18 steps is used. The pir is the rate at which packets of a specific traffic are injected into networks. A pir of 0.001 indicates 0.001 packets per clock cycle are sent by each network node or, in other words, each node sends one packet per 1000 clock cycles. The main network performance characteristics in terms of *average network throughput* (pkt/cycle), *average network latency* (cycle/pkt) and *average network energy cost* (pJ/pkt) are considered in our experiments. The throughput metric refers to the average number of packets successfully received by all network nodes per clock cycle. Latency indicates the average time elapsed between the head flit of a packet injected into the network from the source node and the tail flit of the packet received by the destination node. Energy cost means the average communication energy consumed by a packet traversing from the source to the destination node in networks.

The random bit source traffic at each node generates 5-flit packets with 13 bytes length in each, containing one 1-byte head flit and three 4-byte data flits per packet. All kinds of network nodes have one buffer at each of their directional ports that have 4 flits buffer depth, which allows it to contain at least one flit (head or data). The clock frequency of the whole networks is 1 GHz, meaning 1 nanosecond per clock cycle as the global clock for the experiments. Each simulation is initially run for 3500 clock cycles as sampling time [182] for traffic generation. The first 500 clock cycles of the sampling time are used as warm-up time to stabilise network transient effects [183], which are not included for performance simulations. A further 6500 clock cycles (until 10000 clock cycles) are executed as drain time for ending traffic arriving at their destination nodes. The energy consumed by flit traversal between the source and destination nodes is estimated by the Orion-based energy model in NIRGAM. A trace-based timing method is used for recording the latency performance of the simulations.

4.4.3.2 Result Analysis



(a) Latency of Virtual and Real Honeycomb Networks

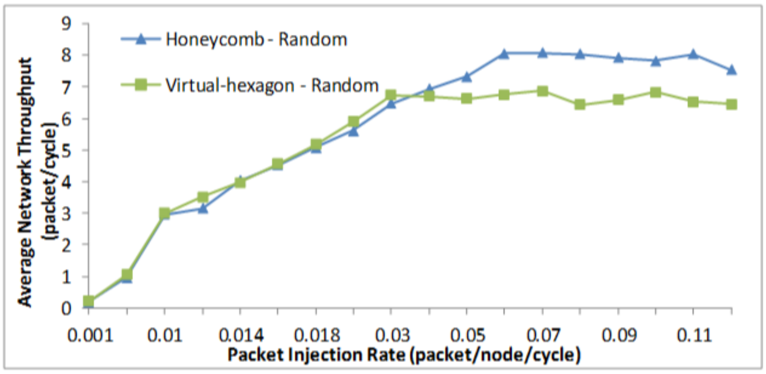


(b) Latency of Virtual and Real Sparse-Octagon Networks

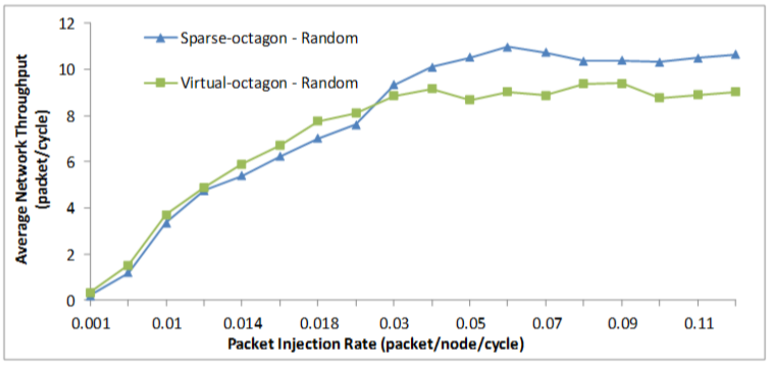
Figure 4.17: Latency Comparisons of Virtual and Real Non-rectangular Networks under Random Uniform Traffic

Figure 4.17 gives the latency performance comparisons of virtual non-rectangular networks over real ones. The green lines in both subfigures are non-rectangular topologies emulated by our time-regulated models while the blue lines are the performance implemented by real networks constructed in the custom way. As observed, both honeycomb and sparse-octagon virtual topologies have precise latency emulations over the range of injection rates before their saturation points. In particular, the virtual honeycomb network emulated by our modelling has an average 5.27% error with the real network before the saturation point (from $pir = 0.001$ to $pir = 0.03$) while the virtual sparse-octagon modelling has an average 4.08% error before its saturation point compared to the real octagon (from $pir = 0.001$ to $pir = 0.02$). This is because of the precise configuration of time-regulated models and affordable network data injection in the mesh network.

But from their saturation points onwards, the emulation results start to give increasing errors from the real topological performance. The reason is that our method intrinsically uses a mesh network with specific routing schemes to emulate non-rectangular routings so that it is easier to cause network congestion, especially when large amounts of traffic saturate the emulated node channels. Hence from their saturation points, the virtual networks will experience heavier traffic congestion, though using larger scale network, than real networks, which leads to worse latency performance on network data transmission. Besides, the saturation points of virtual networks are earlier than the real networks, suggesting more vulnerable ability of virtual topologies to resist heavy traffic scenarios. This is due to the virtual non-rectangular nodes of mesh network experiencing more data contention than the network nodes of real topologies, which saturates the mesh nodes earlier.



(a) Throughput of Virtual and Real Honeycomb Networks



(b) Throughput of Virtual and Real Sparse-Octagon Networks

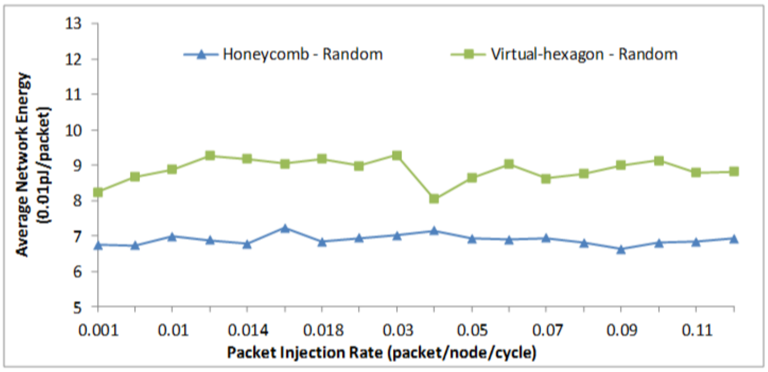
Figure 4.18: Throughput Comparisons of Virtual and Real Non-rectangular Networks under Random Uniform Traffic

Figure 4.18 displays the throughput performance comparisons of virtual and real non-rectangular topologies. Again the green lines in both subfigures indicate virtual topologies and the blue ones for real networks. From both subfigures, the average network throughput of the virtual honeycomb and octagon networks before their saturation points is similar but slightly larger than for their real counterparts. This is because the mesh network scales we used for emulating virtual topologies are larger than the network scales of real topologies, which can hold more data in the networks. Although the specific virtual routings are carefully adjusted in time-regulated models, such throughput differences may be enlarged along with the applied scale of mesh networks for virtual emulation. Specifically, the virtual honeycomb hexagon is applied to a 56-node mesh to emulate a 24-node hexagon, causing an average 4.68% performance error. The virtual octagon is given with a 64-node mesh for a 32-node octagon topology, leading to an average 12.02% throughput error.

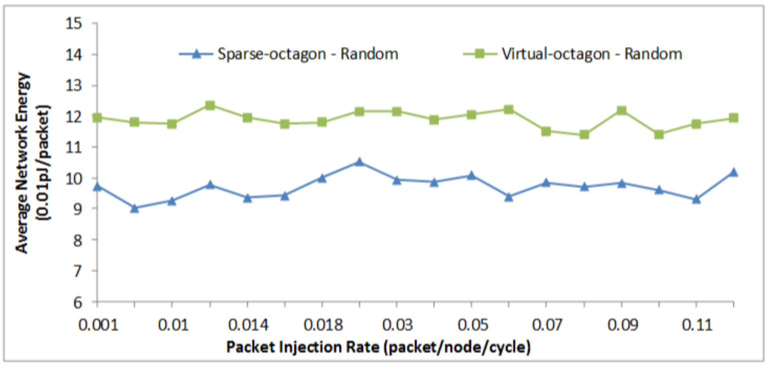
The throughput performance of virtual topologies after their saturation points deviate from the real network performance as the injection rates increases. The actual saturated throughput of virtual non-rectangular topologies is also lower. This is caused by the earlier node saturation of virtual networks than real ones for data transmission. Since the mesh network nodes for virtual topologies are saturated at a lower injection rate than real network nodes, traffic congestion occurs earlier and gives rise to the total available amount of data reaching at a lower level in virtual networks.

Figure 4.19 shows the average packet energy cost by virtual and real topological routings. As seen, energy fluctuates but has little variance, indicating the independence of energy with the change of packet injection rates. This is meaningful since switching activities needed by packets at routers of intermediate nodes from their sources to destinations, as the main consumer of dynamic network energy, will not change with traffic congestion or the time spent at network queues [182].

Moreover, it is observed that in both non-rectangular topologies the virtual emulation networks consume more average energy than real networks (28.71% and 18.27% respectively). This is because the main network energy consumer is the dynamic energy cost of data routings that are largely determined by the average hops of routings. Since the virtual networks are modelled by larger-scale mesh networks and non-rectangular hops are emulated via extra mesh hops with less time spans at routers, the average hops of data routings in virtual networks are obviously higher than in real networks, resulting in the shown extra energy cost. For example, the hexagonal side shown in Figure 4.7(a) ($R0 - R2$) costs 1 hop in the real topology but 2 hops in the virtual mesh network, which causes higher average hops of virtual networks for the same routings.



(a) Energy of Virtual and Real Honeycomb Networks



(b) Energy of Virtual and Real Sparse-Octagon Networks

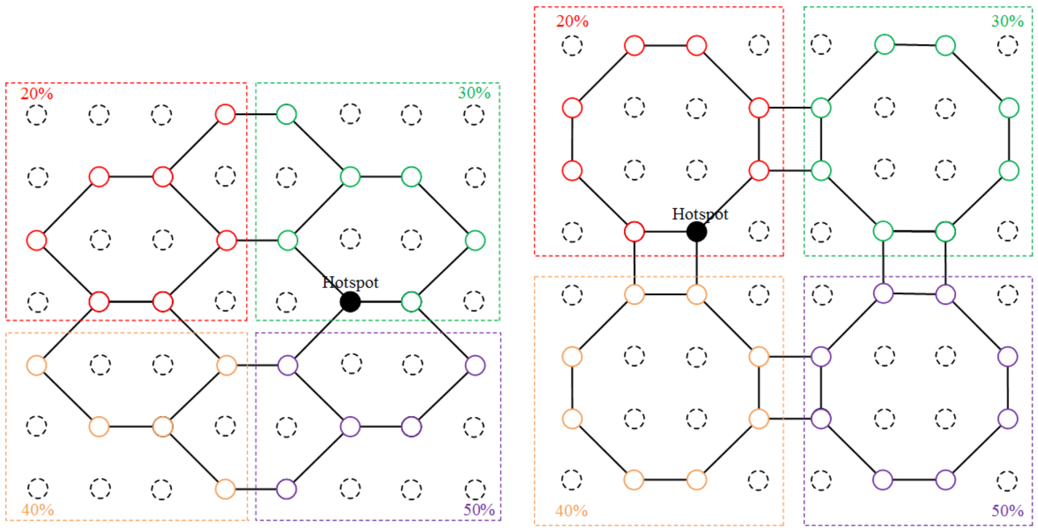
Figure 4.19: Energy Comparisons of Virtual and Real Non-rectangular Networks under Random Uniform Traffic

4.4.4 Synthetic Traffic: Hotspot

4.4.4.1 Experimental Setup

Uniform random traffic is a classic mode for measuring network performance of NoC systems, through it is hardly seen in real scenarios [183]. Hotspot traffic is another typical mode but is similar to many realistic scenarios like communications between microprocessor and memory cores or data transfers via DMA (Direct Memory Access) among various memory cores [163]. In this traffic scenario, if a network node is denoted as a hotspot, it receives extra traffic from other node sources. Other nodes are sending data traffic to random destinations as uniform random traffic. In NIRGAM, if a hotspot node is set as 20% probability, it means the hotspot node will have 20% extra chance to receive 20% additional traffic generated from other nodes.

For our experiments of non-rectangular networks, the hotspot settings are slightly more complicated to inspect the impact on network performance. Figure 4.20 gives the detailed hotspot configuration of both honeycomb hexagon and sparse-octagon networks in terms of virtual and real topologies. One node is denoted as the hotspot node in the two kinds of topologies, respectively, with four different hotspot probabilities. As shown, in both virtual and real non-rectangular networks, red, green, orange and purple regions are created in such topologies, setting the network nodes in those regions to have 20%, 30%, 40% and 50% extra chance to send packets to the hotspot, which also generates 20%, 30%, 40% and 50% extra packets, respectively, when the destination node is the hotspot.



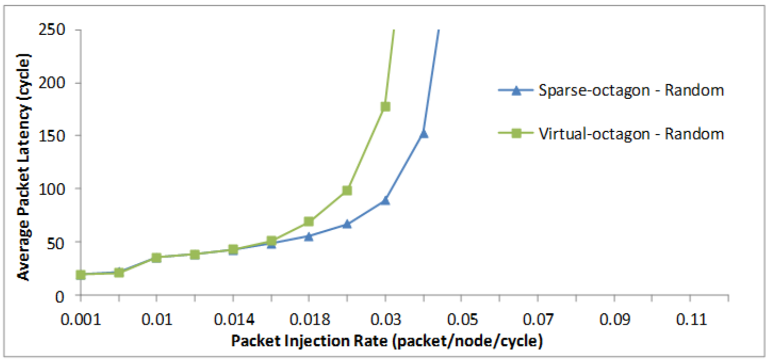
(a) Hotspot Settings for Honeycomb Networks (b) Hotspot Settings for Sparse-Octagon Networks

Figure 4.20: Hotspot Traffic Settings for Virtual and Real Non-rectangular Networks

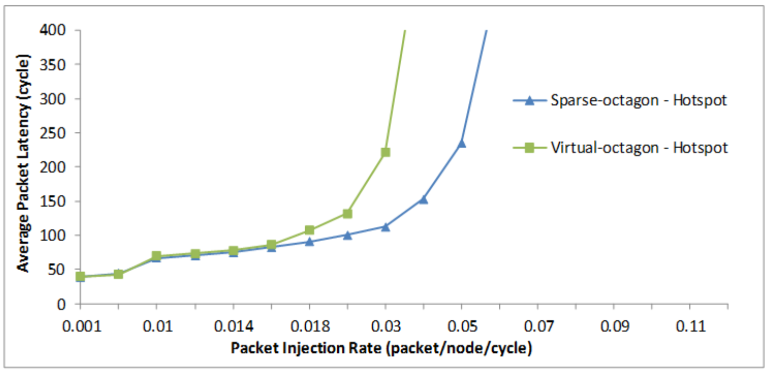
Other experimental settings including buffers, packet and flit length, clock cycle, simulation cycles and range of pir are the same as the uniform random traffic scenario. The performance metrics for measuring virtual and real networks under the hotspot traffic scenario are also the average network latency, throughput and energy cost.

4.4.4.2 Result Analysis

Figure 4.21 demonstrates the latency performance comparisons of virtual and real non-rectangular networks. It is manifest that the performance is similar to the uniform traffic scenario. Before the saturation points of virtual networks, they have close latency costs with only 6.29% and 5.03% errors in the virtual honeycomb hexagon and sparse-octagon from their real counterparts. This indicates the successful and precise emulations of our proposed models and configuration. After the saturation points of virtual networks, the performance deviation rises sharply as the pir increases. Due to more total packets generated and sent to specific node channels around the hotspot node, more severe



(a) Latency of Virtual and Real Honeycomb Networks

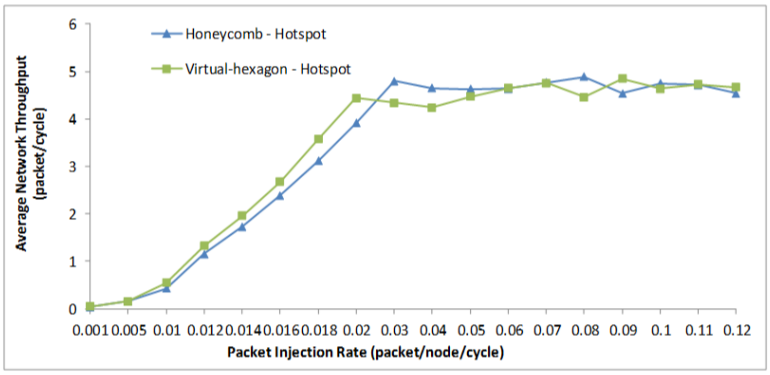


(b) Latency of Virtual and Real Sparse-Octagon Networks

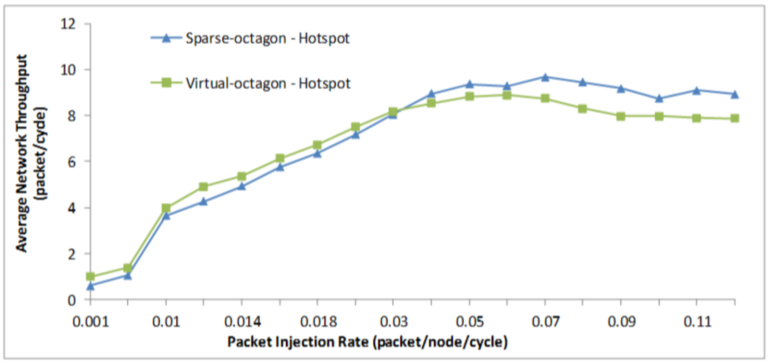
Figure 4.21: Latency Comparisons of Virtual and Real Non-rectangular Networks under Hotspot Traffic

traffic congestion occurred in the virtual topologies, resulting in a sharper increase of data transmission time in mesh-based virtual networks.

Figure 4.22 illustrates the average network throughput obtained by virtual and real non-rectangular networks. Before the saturation points of virtual topological emulations, the throughput performance is close with 11.29% and 13.32% errors, where virtual topologies received slightly more packets than real ones due to the larger mesh network scales used for modelling. The saturation points of virtual networks is earlier at a lower injection rate as the virtual routings saturate certain mesh routing channels earlier than in real networks. While after saturation points, though still lower, the saturated throughput of virtual networks surprisingly approximate the real network throughput, causing more precise performance emulations under hotspot traffic over uniform random traffic (4.32% vs 18.36% in hexagon and 10.33% vs 16.60% in octagon). This may be because under hotspot traffic, the real networks experience heavier traffic congestion around the hotspot node than virtual networks due to smaller network scales and more packets transmitted in networks.



(a) Throughput of Virtual and Real Honeycomb Networks



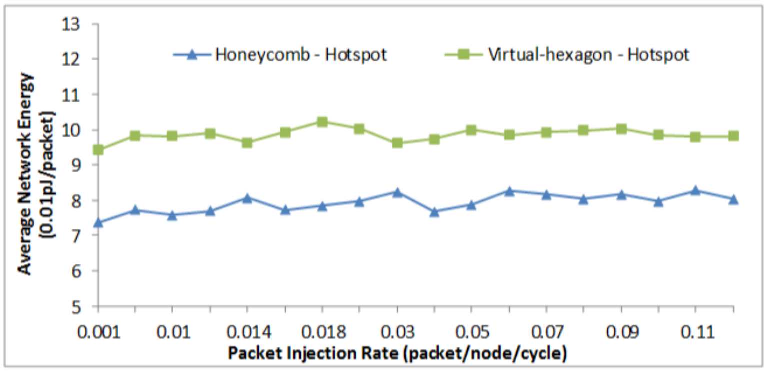
(b) Throughput of Virtual and Real Sparse-Octagon Networks

Figure 4.22: Throughput Comparisons of Virtual and Real Non-rectangular Networks under Hotspot Traffic

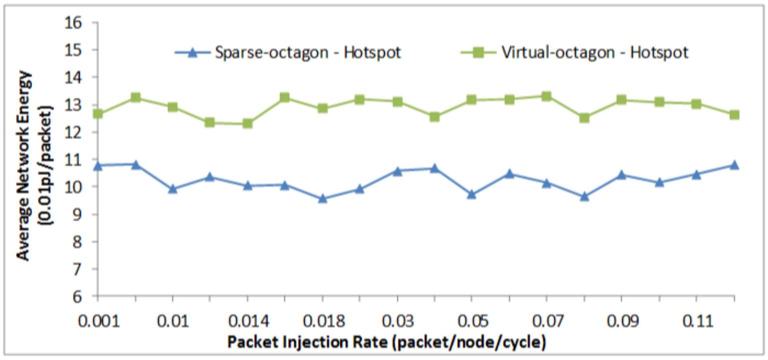
Similar to the uniform random traffic scenario, Figure 4.23 shows that the energy cost by virtual modelling is higher (19.54% and 20.62%) than real networks since the emulation mechanism is not changed and more average routing hops are consumed in virtual topologies. Such performance inaccuracy of energy cost need to be reduced via better approximation of average hops of data routings on virtual networks over real ones. Both virtual and real non-rectangular networks have stable energy cost under a range of different pirs, indicating no obvious impact of heavier traffic congestion under the hotspot traffic scenario on energy performance.

4.4.5 Model Accuracy Discussion

After implementing the comparative experiments of virtual and real non-rectangular topologies, the proposed method using time-regulated models with proper configuration to emulate irregular routings based on a mesh network is discussed in terms of its advantages and limits:



(a) Energy of Virtual and Real Honeycomb Networks



(b) Energy of Virtual and Real Sparse-Octagon Networks

Figure 4.23: Energy Comparisons of Virtual and Real Non-rectangular Networks under Hotspot Traffic

- The proposed model has a limited range wherein precise emulations of network performance in terms of latency and throughput can be achieved. As observed, before the saturation points, virtual topologies emulated by our model can achieve close latency (around 5%) and throughput (around 10%) performance to the real networks, whereas the energy cost has a large deviation (20% $\sim$ 30%) due to the extra average hops of data routings.

- The virtual topologies and routing patterns can be modelled on a mesh network with proper routing schemes and configuration. But such convenience in network construction is at the cost of consuming more network resources and a larger scale of mesh for topological emulations.

- The virtual topological modelling is more vulnerable to traffic congestion, leading to earlier saturation points and worse latency and throughput performance after the network saturates. But under a heavy traffic scenario like hotspot, the larger scale of emulated networks presents a closer performance to the real networks, which suggests a smoother performance degradation to model more precise networks under certain heavy traffic conditions, though at the cost of sharper transmission delays.

It is worth pointing out that the non-rectangular topological modelling is just part of the functionality our proposed emulation method can provide. Irregular and other application-specific routing patterns are also achievable by using our time-regulated models. The errors and deviations exposed in performance simulations of our virtual networks can also be avoided or even remedied in other designs. For example, if applications require packet injection rates in a proper range and specific routing patterns are implemented, virtual topologies modelled by our method on a mesh network can have equal (or close) routing hops to the real architectures. Thus, efficient emulations and accurate performance in terms of latency, throughput and energy can both be achieved by our method.

4.4.6 Specific Application: MPEG-4 Decoder

To evaluate the potential of our proposed method for real multimedia applications, an MPEG-4 decoder was mapped onto a custom irregular network and on a mesh network with virtual emulations of the same irregular routing with time-regulated models. Both networks are constructed based on the framework of a 9 by 9 Mesh. Buffer depth and width are set equal to the previous synthetic traffic experiments. Virtual channel allocation is also configured to equalise the setting of specific routing experiments. The source routing algorithm is used for regulating routing paths on both networks. The timing and energy performance of the decoding process on both networks are compared in the NIRGAM simulator to validate the accuracy of the proposed method.

4.4.6.1 Software Implementation

Figure 4.24 show the modules of an MPEG-4 decoder constructed in the NIRGAM simulator. The numbers shown at the links of node pairs indicate the total tasks required for mutual communications to decode a 1-second QCIF MPEG-4 video clip with 25fps frame rate. The software implementation of MPEG-4 decoder is extracted from an online open source Xvid codec [184], classifying specific IP cores which can be attached to network nodes for the NIRGAM simulation. The modules given in the figure are split from the original decoder source and behave as hardware modules instead of software functions, so that its generated application placement is a baseline suitable for further experiments and investigation.

In the decoder application, the decoding process is split into 8 different modules with certain communications between each other that are placed onto 8 independent network nodes. Each module takes charge of one particular part of tasks in decoding process

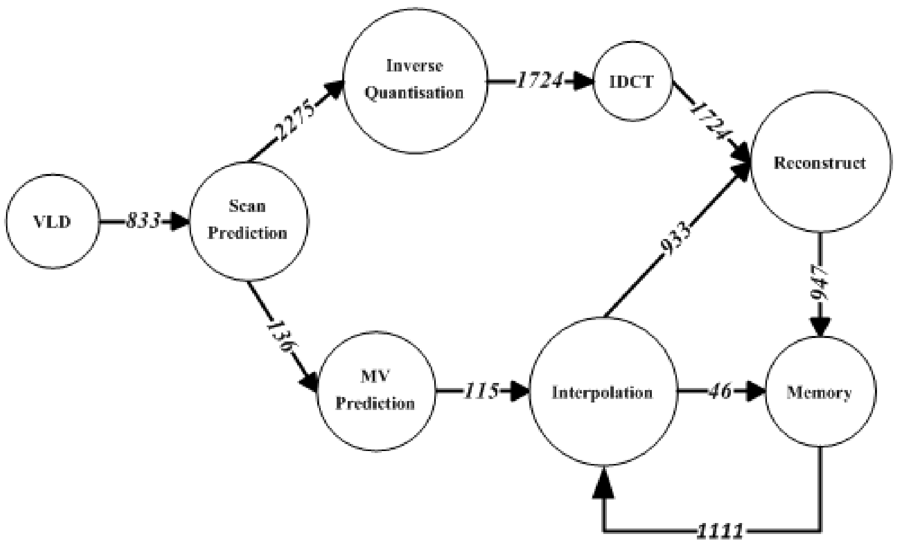


Figure 4.24: MPEG-4 Decoder

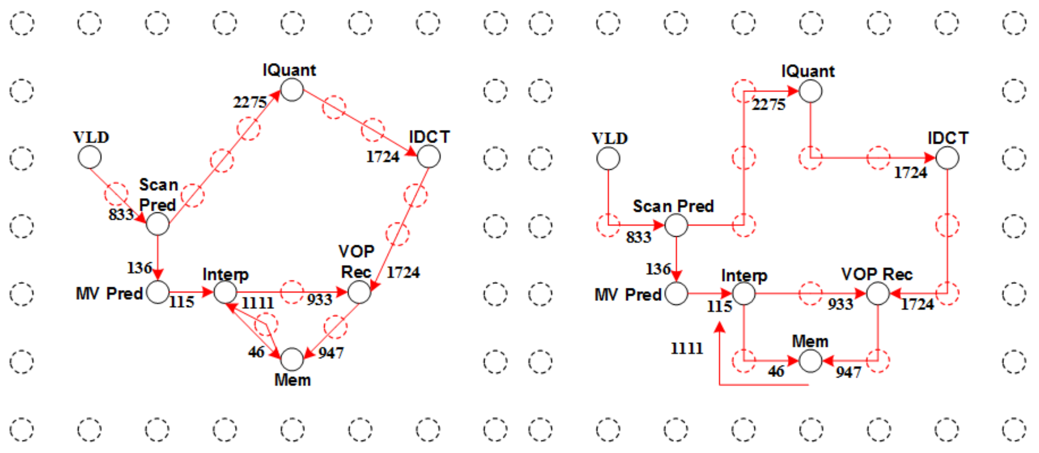
and have sequential input and output communications with other modules via network routers. All the mutual communications require correspondent network channels for data transmission in the network. The initial input of the decoder is the encoded video data stream sent to the Variable Length Decoder (VLD) module initially and the output is a set of decoded and restored YUV data generated at the Reconstruct module.

4.4.6.2 Experimental Setup

Although not normally used today, the use of 45° interconnects has been proposed previously [185]. This example shows how such interconnects could be modelled in NIRGAM. A real network customised in a conventional way and a virtual topology emulated by our method are both used to implement the MPEG-4 multimedia application. The decoding process starts by sending the encoded video bitstream from the VLD core. Figure 4.25 gives the module placement of the MPEG-4 decoder on custom network directly and on mesh network with virtual modelling of irregular routings. Since the bandwidth requirements of mutual communications at each pair exceed the available bandwidth between node channels, intermediate nodes are introduced on both real and virtual networks for the irregular routing of MPEG-4 decoding, offering extra buffers at each node router and prolonged routing path to avoid packet drop. Black solid-line circles indicate modules of the decoder, red solid lines indicating routing path for the decoding process, and red dash-line circles suggesting the intermediate nodes used in both network topologies for successful data transmission.

Specifically in Figure 4.25(a), **a custom network** is constructed based on the 9 by 9 mesh framework. Seven module pairs of the decoder have multi-hop routings which are *[VLD-Scan] (2 hops), [Scan-IQuant] (4 hops), [IQuant-IDCT] (3 hops), [IDCT-VOP Rec] (3 hops), [Interp-VOP Rec] (2 hops), [VOP Rec-Mem] (2 hops) and [Interp-Mem] (2 hops)*, respectively, attaching onto the normal mesh network nodes. It is noteworthy that the module pair [Interp-Mem] in the custom network has bi-directional communications that transmit data with different bandwidth requirements. This difference makes one direction ([Mem] to [Interp]) cost 2 hops and the other directional communication ([Interp] to [Mem]) cost 1 hop for successful task transmission.

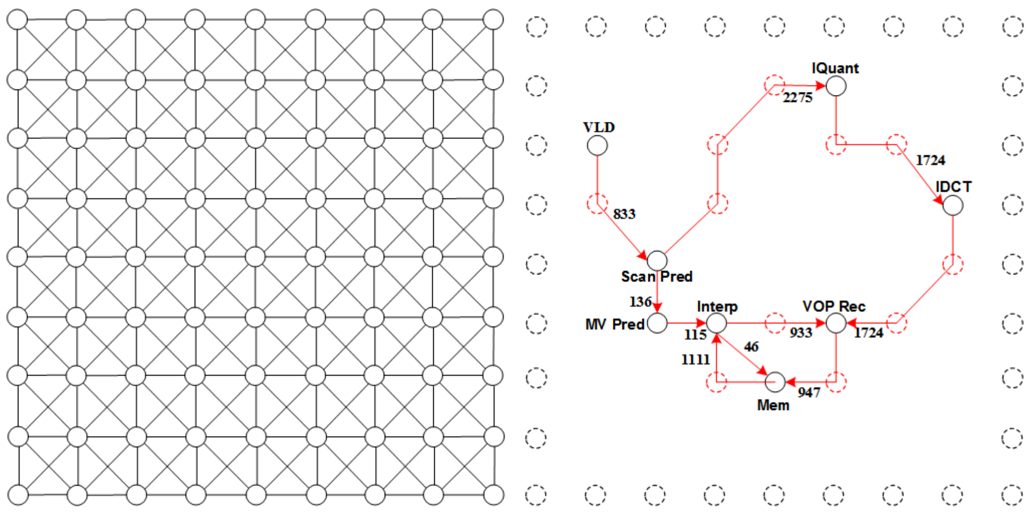
The MPEG-4 decoder is also mapped onto **a mesh network with virtual emulations** to the irregular routings using time-regulated models. Red dash-line circles in Figure 4.25(b) represent the nodes with the time-regulated model in the mesh. The configuration of those models are carefully adjusted to make data routings on red solid lines on the mesh network equal to on the custom network for precisely emulating the irregular routings of MPEG-4 decoder. The number of red dash-line circles are intentionally set to the same as used in the custom network to approximate the average data hops in the network for accurate energy modelling. It is notable that the bi-directional communications at module pair [Interp-Mem] cannot be modelled with different hops as in the custom network. This is because the virtual emulations route data on the mesh network, which follows the Manhattan geometric rules and can only allocate 2 hops to both directional links. This characteristic leads to one extra hop consumed by the same bi-directional links in the virtual network than in the custom network.



(a) Irregular Placement on a Custom Network (b) Virtual Irregular placement on Mesh

Figure 4.25: Module placement of MPEG-4 Decoder for Irregular Routing

To illustrate how the custom network shown in Figure 4.25(a) is implemented in NIRGAM, a customised Mesh with 8-neighbour homogeneous nodes is firstly designed as shown in Figure 4.26(a). Such network nodes have 8 instead of the default 4 directional links to neighbouring nodes. Specifically, four new directional links (Northwest, Northeast, Southwest and Southeast) are added to the node router. Since this custom network is for comparative experiments in order to test our proposed method for real-world multimedia applications, the network is synchronous and all the settings to these 8 link channels are the same as the default Mesh network. Hence, cores from the graph in Figure 4.24 are mapped onto this custom network and intermediate hops between those module pairs are also needed in order to meet the communication requirements. The irregular placement given in Figure 4.25(a) are finally implemented as shown in Figure 4.26(b). Module pairs of *[VLD-Scan], [Scan-IQuant], [IQuant-IDCT], [IDCT-VOP Rec], and [Interp-Mem]* are implemented by using one of the four new directional links the network node routers have been added. All other unused Mesh nodes and links from the decoder mapping are disconnected to form the specific irregular topology.



(a) Customised 8-neighbour Mesh Network (b) Implementation of the Irregular Placement

Figure 4.26: Implementation of the Irregular Placement of MPEG-4 Decoder

4.4.6.3 Result Analysis

Table 4.3 lists the performance comparisons of MPEG-4 decoding on the custom and Mesh networks. It shows that by using our virtual modelling, the performance of irregular routings on a mesh is emulated with 0.13% timing error and 3.07% energy error compared to the performance on the custom network.

Table 4.3: Performance Comparisons of MPEG-4 Decoding

Performance	Mesh with Virtual Routings	Custom Network	Error (%)
Time (μs)	4131.795	4137.157	0.13
Energy (μJ)	154.59	149,982	3.07

The accurate timing emulation of the MPEG-4 decoder given by the proposed method is due to the precise timing configuration of the time-regulated models for irregular routings. The 3% error of energy performance in the Mesh-based virtual irregular routings is due to the 1 extra hop between the unidirectional communications at [Interp-Rec] module pair.

4.4.7 Superiority of Our Method for Topological Modelling

Consequently, we have developed a novel method to resolve the conflict between design performance and design productivity by considering both the module reusability and high performance demands. The method uses the NIRGAM simulator as the backbone framework, offering a fast and easy way to emulate virtual non-rectangular topologies and irregular routing patterns based on a regular mesh network. The modules for design automation of mesh network in the simulator are fully reused for other specific topologies. Configurable time-regulated models are designed and attached on mesh network nodes that adjust data transmission time at nodes pairs to form non-rectangular and irregular routing geometries with precise timing and energy performance. In our method, efficient topological emulations with precise performance simulations for many non-rectangular networks and irregular geometries are achieved by reusable modules, which combines merits of both former methods while overcoming their shortages. The conflict between performance and productivity is alleviated by our method, while module reuse and accurate performance modelling are both considered.

Our method does not construct the *real* topologies in the simulator with unique and specially designed modules. Instead, the topological geometries are investigated and emulated *virtually* by attaching the time-regulated model with different settings onto a mesh network, which makes full use of the reusable modules that a simulator has integrated for its default mesh network. Traditionally, modern simulators often provide the platform and functionality for researchers to customise their specific network topologies, including self-designed characteristic modules, structures and routing schemes. The development process may be time-costly and the unique modules may be not reusable. In our method, the reusability and modularity of applied modules is guaranteed by the pre-designed mesh network. The heterogeneity required for specific topologies is only presented by the time-regulated models with certain configuration, which obviously saves time for custom module designs.

To explicitly demonstrate the superiority, a list of characteristic modules, simulator modifications and environmental settings needed in designing custom topologies is given to compare our method and the custom way. To make the comparison fair, all the generation processes required by the two methods are given in the NIRGAM simulator. The comparative results are shown in Table 4.4

Table 4.4: Comparison of Topology Design in Custom Way and by Our Method

Topology Designs need:	Custom Way	Our Method
Learn characteristics of each topology	✓	✓
Design routers	✓	×
Design network interfaces	✓	×
Design I/O ports	✓	×
Design crossbars	✓	×
Develop routing schemes	✓	✓
Configure time-regulated models	×	✓

From the table, it is observed that our method needs much less design than THE custom way in NIRGAM to simulate a network topology. Several modules like routers, I/O ports and network interfaces need to be customised specifically for each different topology. But in our method, these designs can be completely saved since all these relevant modules in the mesh topology are reused for different topologies. Such superiority in design time savings can be amplified if multiple topologies are desired to be implemented for performance simulation.

With the thorough study of candidate topologies, the time-regulated models are carefully set in our method to force the mesh network routing data to be as similar to certain geometric patterns, producing precise transmission time as routing on the specific real topologies. The energy performance of such emulated routings is highly relevant with the average data hops they consume, which can be precisely approximated to the cost by real topologies as proven in former experiments. It is noteworthy that although the proposed method has superiority in efficiently emulating and modelling specific topological routing performance, limits like reduced range of applicable packet injection rate and vulnerable tolerance to traffic congestion, which are discussed before, require cautious use of our model.

4.5 Summary

In this chapter, we have proposed a novel method, instead of specifically constructing each custom network architecture, using time-regulated models to emulate non-rectangular and irregular NoC topologies on a regular mesh network in the NIRGAM simulator. This provides a rapidly generated, easy-to-implement approach to emulate multiple topologies and routing patterns with efficient module reuse and precise performance modelling, which contributes to design space exploration of system-level modelling step of current NoC design methodology as shown in Figure 1.2. This method is practicable and needed at design-time for exploring different NoC topologies. Honeycomb hexagon, Sparse-octagon and two irregular routings are developed by our method to measure the performance in terms of latency, throughput and energy. Results have verified the functionality of our method. Moreover, an MPEG-4 decoder is also implemented with special irregular routings to evaluate the potential of our method for modelling real-world multimedia applications.

Apart from the desirable experimental results, problems and limits of our method are exposed as well. Potential remedies and solutions to those drawbacks are required. **Firstly** the method is limited by fixed network degree of the mesh network. The honeycomb and sparse-octagon virtual emulations both have lower network degree than the mesh, which thus enables precise modelling. This limit may be alleviated via using a high-degree network as the backbone for virtual emulations.

Secondly the applied range of packet injection rates is limited in the virtual topologies. After the saturation points of the given virtual networks, which are earlier than real networks due to heavier congestion at virtual nodes, the virtual routings may fail to precisely emulate the network characteristics of real networks. This is due to the heavier traffic loads caused by the model configuration and routing schemes. Hence, potential remedies would be to design more advanced, load-balancing routing schemes and time configuration to enlarge the applied range. It is adversely deducible that the virtual modelling offers narrowed applied range for precise performance that is less likely to saturate the target networks than real ones since the reduced range of unsaturated as well as desirable network conditions will be considered in virtual topologies. This limit of our model, however, restricts a more cautious usage of the virtual networks for normal operations, which is not bad for those applications that require unsaturated implementations.

Thirdly the average data routing hops and network scales of virtual networks modelled by our method are higher than real ones, leading to more (inaccurate) energy and more network resource cost. A promising potential solution to such a problem is to use similar network resources (nodes) to model virtual networks with close average hops. For example, the virtual 24-node honeycomb and 32-node sparse-octagon can be emulated their brick shapes as shown in Figure 4.3 on the mesh network, rendering to the drops of network scales from 56 and 64 nodes to 28 (7 by 4) and 32 (8 by 4) nodes, respectively. Such mesh networks can also cause more close average routing hops for virtual modelling, which alleviates the energy error.

Finally, virtual modelling and emulations are more vulnerable to traffic congestion, which is a hard-core limit of our method that is hardly solved. It is because the applied range of mesh nodes for virtual emulations are always narrower than real networks. So the traffic load resistance is always limited in the virtual modelling. A possible way to alleviate this situation would be to develop load-balancing routing algorithms and routing schemes. But this will introduce extra performance errors. A generalised balance between the routing applicability and error control is hard to find, though it is still possible.

Chapter 5

Application Mapping and Performance Prediction

Task mappings of various applications to network IP cores with certain interconnections is a significant design stage in the Network-on-Chip (NoC) design methodology for achieving desirable system performance. Normally one application will be split into several tasks and mapped onto different nodes of an on-chip network. As the design complexity of modern applications increases, efficient decision makings on proper task mappings and network architectures are more desirable to produce efficient design cycles and to alleviate the ever-severe time-to-market pressure. Hence, to balance the design productivity and design performance of NoC systems, application mapping techniques at early design stages have become a hot research topic.

In this chapter, developing a novel mapping strategy that balances design performance and design productivity is our research focus. We expect the proposed strategy can rapidly decide the optimal task mapping on various network topologies. Moreover, obtaining a precise prediction of certain performance metrics (communication energy and transmission latency) of a mapping for further design stages is expected to shorten the design cycles. The reason to develop such a mapping strategy is not only because of the impact application mapping techniques may have on the optimisation of network system performance, but also because of the effective and useful performance prediction such strategies may offer at the early design stages for high design efficiency. Besides, less requirements in design cycles and more precise performance prediction can also lower the design cost of prototyping of complex NoC systems.

5.1 Introduction and Necessity

In modern application-specific NoC designs, a popular NoC design flow has been widely used (as given in Figure 5.1) [15]. An *application* is specified as a set of computational tasks with certain communications with each other, which is known as an application *task graph*. IP functional cores in NoC network nodes can process a subset of these tasks. Hence, the initial step of the design flow is to select a set of available network nodes and properly allocate the computational tasks to them. This gives rise to the *mapped node graph* in the design flow in which the network nodes and communication bandwidths needed by the channels between node pairs are labelled as nodes and edges of the network topology, respectively. At the *mapping* stage, numerous techniques have been developed to map the task graph onto a network topology graph with the fulfilment of mapping objectives to specific design requirements. The mapped graph is passed to the *routing* and *scheduling* stages to form and establish the target NoC network.

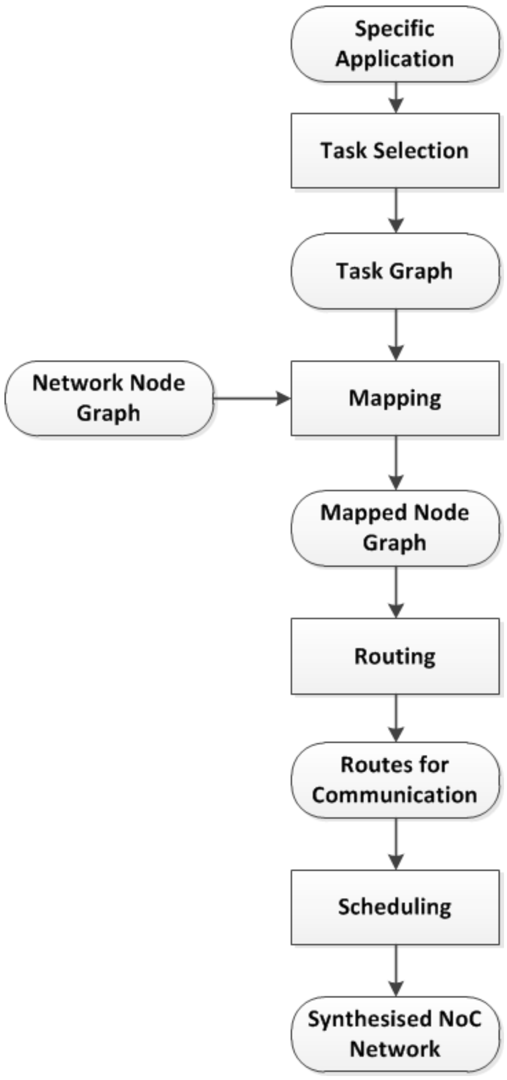


Figure 5.1: Design Flow of Most Application-specific NoCs

From the design flow, it is clear that improvement and optimisation in developing task mappings to node interconnection networks at the mapping stage is essential to improve the NoC design performance, which requires advanced uses of modern EDA techniques to determine optimal task mappings. But to serve our objectives, improving design efficiency of NoC systems at early stages is also significant. For this reason, we expect to efficiently produce optimal task mappings with precise and reliable performance predictions on a range of different network topologies. The models for virtual topological emulations introduced in the last chapter can be combined with such an ideal mapping technique to improve the efficiency of NoC design methodology from *task selection* step to *routing* step.

However, as more complicated tasks being processed in modern applications, total feasible task mappings rise exponentially along with the processing core number, which costs longer execution time to search the optimal mapping. This searching time may become unacceptably long since the optimisation of application mapping problems is an instance of constrained quadratic assignment problem, also known as an NP-hard problem [91] [92] [93]. In this case, the difficulty of achieving optimal mappings increases and the execution efficiency of mapping techniques to search optimal mappings decreases, indicating a reduction in both design productivity and design performance of task mappings to complicated NoC applications. Besides, a generalised optimal task mapping suitable for all applications is hard to produce since their various and specific design requirements may not be optimised concurrently or even contradict each other.

For this reason, many task mapping techniques have been developed to search optimal mappings of specific applications with certain performance metrics such as communication energy, transmission time, link bandwidth and traffic workloads [15]. Yet few of these previous mappings have balanced their mapping result accuracy and execution efficiency. Some of them, like evolutionary computing techniques, trade off execution efficiency with increased iterations for refining mapping performance, while some constructive heuristics have a fast mapping construction in reduced execution periods at the cost of acquiring near-optimal results without improvement. For early-stage performance prediction, a mapping technique with a better balance between execution efficiency and result accuracy is expected to improve the design productivity while keeping the design performance. Moreover, most previous mappings are designed on a mesh NoC architecture due to the easy problem formulation and structural implementation, despite the fact that other NoC architectures can also process applications with desirable performance like in [173] and last chapter.

In Chapter 2, we found that the mapping techniques that can apply to various NoC architectures and have a good balance between execution efficiency and performance accuracy are rare. Motivated by this, we have proposed a novel method to fill the gap, developing a mapping technique that is extensible onto different regular NoC topologies and produces globally optimal mappings with high execution efficiency and precisely calculated performance for early-stage prediction. The major features included in our mapping technique are listed below:

- The objective is to minimise the cost of communication energy and transmission latency under link bandwidth and execution time constraints. The reason is considering both metrics instead of either single one can better evaluate their impact on system performance. Moreover, considering either metric only may lead to inapplicable optimal results. For example, a mapping minimising energy cost may place tasks in a compact area causing high traffic congestion while a mapping minimising time cost may average network traffic loads and place tasks widely costing high communication energy.
- A Non-Linear Programming (NLP) based technique is used in our problem formulation to improve the method's extensibility. Instead of depicting task connections based on architectural features like node directional neighbours, a coordinate system with certain coefficients representing the node intervals is developed, which is easily extended for other regular tile-based NoCs in which the node interrelations can be described by coordinates.
- A modified branch and bound (BB) algorithm is used to offer global optimality to generated mappings with increased execution efficiency. The BB algorithm explores the search space that covers all possible solutions and is compatible with NLP based problem formulation. Our modification to original BB algorithm improves its bounding mechanism by trimming more unlikely tree branches of the solution set at early levels to accelerate the searching speed.
- Our proposed method develops a fast and efficient solution of producing optimal task mappings with precise early-stage performance predictions. State-of-the-art energy and timing models are modified to calculate accurate mapping performance. An event-driven, cycle-accurate simulator, NIRGAM [25] [163], is integrated to provide baseline metrics in the models for problem formulation. It also provides performance simulations to check the accuracy of calculated results.

The organisation of this chapter is as follows: Basic network knowledge for the proposed mapping technique is introduced firstly. The next section details the problem formation and extensibility of our method, followed by the modified BB mapping algorithm and process of performance prediction. Experiments are implemented and analysed finally to validate our method.

5.2 Proposed Mapping Method and Performance Prediction

In this section, our mapping technique is introduced. General knowledge of NoC architectures and performance models used in our design is initially given, followed by the details of NLP-based problem formulation. Then the extension of our formulation technique onto various NoC architectures is elaborated. Modification to a Branch and Bound mapping algorithm for increasing its searching efficiency is explained next. Finally the design methodology to generate accurate performance prediction by our method is illustrated.

5.2.1 Preliminaries

5.2.1.1 The Architecture

Topology:

In this section, an n-by-n 2D-mesh NoC topology is used to implement task mappings due to its desirable properties like structural regularity, concurrent data transmission and modular reusability [18]. As shown in Figure 5.2, an example 3-by-3 mesh network is presented with 9 network tiles/nodes interconnected, where each tile consists of one IP core, one buffer and one router switch. This NoC architecture is already given in Figure 3.3, which is reproduced in this chapter for convenient explanations. Particularly, since the coordinate system used in our method can be extended to other specific network topologies, the Mesh-based topology is not the only platform suitable for our mapping method. Other regular, tile-based topologies may also be suitable for the proposed mathematical description if their node connections can be described by coordinates.

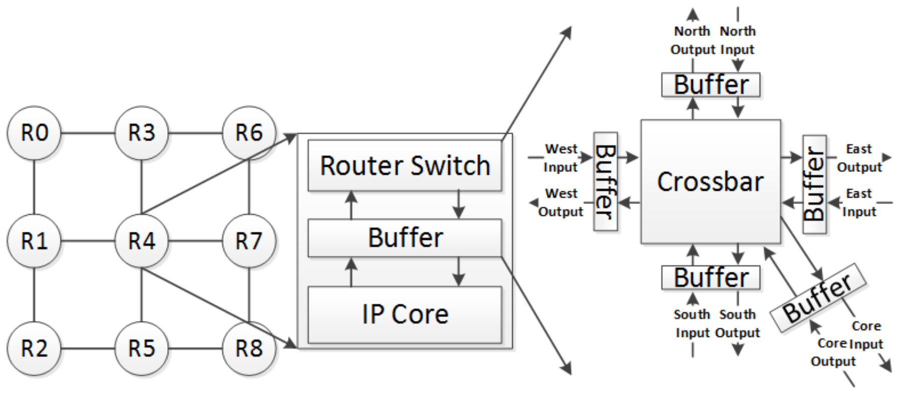


Figure 5.2: Typical Structure of a 2D-Mesh Network Tile and Its Router Switch

Switch:

In a network node/tile, the router switch normally has five directional links connecting to four neighbouring tiles and a local IP core via channels. Each channel is composed of two opposite one-directional physical links with one end connecting to neighbouring nodes and the other connecting to a crossbar via a small buffer. The buffers are designed for registering waiting data since the contention of multiple data packets for occupying one specific channel concurrently may cause packet drops. The physical link channels can be multiply utilised by dividing into several virtual channels in the meantime. The crossbar arbitrates which directional output to send a data packet to by reading the destination or source information contained in the header.

Wormhole Routing:
The wormhole-based routing technique [36] has been used for NoC architectures since it is well-coupled with the reduced buffering resources and strict timing requirements desired by classic NoC applications [116]. In this routing technique, data packets are divided into smaller units called flits (flow control units) and pipelined through the network. All the routing and destination information are contained by the head flit which is arbitrated by intermediate node routers, followed by other flits of the packet via the same node. If the head flit is blocked or waiting at one node due to traffic congestion or data contention, other trailing flits will stay at their local nodes instead of continuing to the blocked node, which substantially reduces the buffering space requirement at each node. This characteristic makes wormhole routing suitable for NoC architectures.

Deadlock- and Livelock-free Routing:
Deadlock-free and Livelock-free routings are critical in NoC architectures to prevent severe performance impairment or even failure of the network traffic. As per [48] and [116], two general classes of wormhole-based routing techniques have been presented which are deterministic routings and adaptive routings. All deterministic routing algorithms are livelock-free [51]. More details are given in Chapter 2. For the deadlock-free algorithms in our mapping technique, a static XY deterministic routing algorithm [40] is used for the mesh network and as the basis for other NoC tile-based architectures for the following reasons:

- It is easy to design the structure and simple to implement with less consumption of network resources compared to adaptive algorithms.
- It is deadlock-free and livelock-free [51].
- Its routing mechanism avoids the reordering of arrived data flits at the destination node.
- It needs very limited buffering space at intermediate routers, which substantially reduces the possibility of resource overheads [108].

5.2.1.2 Energy and Timing Models

The trade-off cost of communication energy and transmission time is the major objective in our mapping problem. Extracting accurate energy and timing models is essential for precise performance predictions, which necessitates deriving the models accurately before problem formulation. For the energy model, Ye *et al.* [186] firstly presented a *bit energy* performance metric, which is defined as the energy cost when one single bit of data travels through a router.

$$E_{bit} = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} \tag{5.1}$$

where E_{bit} refers to the bit energy. $E_{S_{bit}}$, $E_{B_{bit}}$ and $E_{W_{bit}}$ represent the energy consumed by a single bit of data routing through the router switch, buffers at channel ports and interconnection switching wires, respectively. The energy model shown in Equation 5.1 has been modified in [116], adding a part of energy cost on channel links between network node pairs. The new energy model of a single bit of data sent between a node pair is given in Equation 5.2:

$$E_{bit} = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} + E_{L_{bit}} \tag{5.2}$$

where $E_{L_{bit}}$ indicates the energy consumed by a single bit of data traversing channel links between an adjacent node pair. At this step, Hu *et al.* further modified the new model in [116] to neglect the parts of energy cost by the buffering ($E_{B_{bit}}$) and internal switching wires ($E_{W_{bit}}$) due to their minor influence. The simplified energy model thereafter was widely used in many other works. Unlike the simplification, our energy model integrates one more part to the complete model shown in Equation 5.2 to improve the result accuracy. It is based on a modified energy model derived from a popular power model Orion [87] [157], depicting the communication energy cost between IP core and router switch:

$$\begin{cases} E_{bit} = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} + E_{L_{bit}} + E_{src-C_{bit}} + E_{dst-C_{bit}} \\ \quad = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} + E_{L_{bit}} + 2 \times E_{C_{bit}} \\ E_{C_{bit}} = E_{src-C_{bit}} = E_{dst-C_{bit}} \end{cases} \tag{5.3}$$

where E_{Cbit}, $E_{src-Cbit}$ and $E_{dst-Cbit}$ represent the energy consumed between the IP core and switch, by a single bit of data traversing from the IP core to the crossbar buffer at the router switch of the source node, and from the crossbar buffer at the router switch of the destination node to IP core, respectively.

Since each time a bit of data routing would start from the core of the source node and end at the core of the destination node, the energy cost between the IP core and the switch at both the source and destination nodes are equal. Moreover, it can be observed that each time a bit of data traverse through a router switch, $E_{S_{bit}}$, $E_{B_{bit}}$ and $E_{W_{bit}}$ are always included and computed together. Besides, each data routing will always and only contain one $E_{src-Cbit}$ and one $(E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}})$ at the source node, one $E_{dst-Cbit}$ at the destination node as data will always route the path from the IP core to the router switch at the source node, outputting from the source node switch and finally arriving at the IP core buffer at the destination node. Hence, the term in Equation 5.3 can be merged to simplify the equation:

$$\begin{cases} E_{S'_{bit}} = E_{S_{bit}} + E_{B_{bit}} + E_{W_{bit}} \\ E_{S-D_{bit}} = (E_{src-S_{bit}} + E_{src-B_{bit}} + E_{src-W_{bit}}) + (E_{src-C_{bit}} + E_{dst-C_{bit}}) \\ = E_{src-S'_{bit}} + 2 \times E_{C_{bit}} \end{cases} \tag{5.4}$$

where $E_{S'_{bit}}$ refers to the total energy consumed by data passing through a router switch. $E_{S-D_{bit}}$ indicates the constant energy consumed by a data transmission at the source and destination nodes, which contains the energy cost for routing from the IP core to the switch of the source node $((E_{src-C_{bit}})$, outputting from the router switch of the source code $(E_{src-S'_{bit}} = (E_{src-S_{bit}} + E_{src-B_{bit}} + E_{src-W_{bit}}))$, and to the IP core from the router switch at the destination node $(E_{dst-C_{bit}})$, respectively. Based on the simplification of Equation 5.4, the new energy model for sending a single bit of data from node n_i to n_j has been derived from Equation 5.3:

$$\begin{aligned} E_{bit}^{n_i,n_j} &= (hop_{i,j} \times E_{S'_{bit}} + hop_{i,j} \times E_{L_{bit}}) + E_{S-D_{bit}} \\ &= hop_{i,j} \times (E_{S'_{bit}} + E_{L_{bit}}) + E_{S-D_{bit}} \end{aligned} \tag{5.5}$$

where $hop_{i,j}$ is the hop count between node i and j (switch i and j). It is noteworthy that Equation 5.5 is a linear function of the variable $hop_{i,j}$ and constants $E_{S'_{bit}}$, $E_{L_{bit}}$ and $E_{S-D_{bit}}$. The constant polynomials can be further simplified, which results in **the final form of our energy model**:

$$\begin{cases} E_{bit}^{n_i,n_j} = hop_{i,j} \times E_A + E_B \\ E_A = (E_{S'_{bit}} + E_{L_{bit}}) \\ E_B = E_{S-D_{bit}} \end{cases} \tag{5.6}$$

Providing the constant values are already known (the values of baseline references are achieved from the simulation of a single data bit routing through NoC networks in NIRGAM), the total communication energy cost can be optimised regardless of the data routing model. By using Equation 5.6, the energy minimisation problem can be replaced by determining the optimal Manhattan hopping distance between the data routing path from node n_i to n_j.

The timing model can be derived in the same way since in all energy cost parts the time is always consumed in the same way. Similar to E_A and E_B, corresponding timing constant values of T_A and T_B can also be obtained from NIRGAM as the baseline references. Hence the proposed timing model is:

$$T_{bit}^{n_i,n_j} = hop_{i,j} \times T_A + T_B \tag{5.7}$$

5.2.2 Problem Formulation of NLP Based Mapping

In this section, we present details of a Non-Linear Programming (NLP) based formulation for application mapping problems. Several critical definitions about the task graph and network graph are given first, followed by other variable configurations needed for the problem formulation.

5.2.2.1 Problem Definition

Similar to many other mapping problem formulations like in [119], several critical definitions are needed to formulate our mapping problem.

An application will be divided into a set of tasks according to their interconnections, which forms a *task graph* to be mapped onto NoC architectures. The task graph is defined below:

Definition 1: A task graph for an application is a *directed* graph, $\top = G(V, E)$, where each vertex $v_i \in V$ represents a selected IP task. A directed edge $e_{i,j} \in E$ represents the communication from vertices v_i to v_j. For any vertex pair v_i, v_j, there can be two directional arcs at most with each arc dedicated to one direction. For each directed edge $e_{i,j}$, there are the following properties:

- $comm(e_{i,j})$ denotes the arc volume from tasks v_i to v_j, representing the communication volume (data bits) from v_i to v_j.
- $b(e_{i,j})$ denotes the weight of edge $e_{i,j}$, representing the minimum bandwidth (data bits per second) of the communication from tasks v_i to v_j required by the application tasks to fulfil performance constraints.

Definition 2: An NoC architecture graph for an application is a *directed* graph, $\top' = G(N, L)$, where each vertex $n_i \in N$ represents a node in the network. A directed edge $l_{i,j} \in L$ represents the link routing communication from vertices n_i to n_j. For any vertex pair n_i, n_j, there can be two directional arcs at most with each arc dedicated to one direction. For each directed edge $l_{i,j}$, there are following properties:

- $P_{i,j}$ denotes the set of potential minimal routing paths from nodes n_i to n_j, representing the communication volume (data bits) from n_i to n_j. $\forall p_{i,j} \in P_{i,j}$, $Link(p_{i,j})$ denotes the channel links used by a specific $p_{i,j}$.
- $bw(l_{i,j})$ denotes the weight of edge $l_{i,j}$, representing the bandwidth (data bits per second) available across the link channel from nodes n_i to n_j in the NoC architecture.
- $e(l_{i,j})$ denotes the edge energy cost, representing the communication energy cost (joules) of sending one bit of data from nodes n_i to n_j, which is $E_{bit}^{n_i,n_j}$.
- $t(l_{i,j})$ denotes the edge timing cost, representing the transmitting time cost (seconds) of sending one bit of data from nodes n_i to n_j, which is $T_{bit}^{n_i,n_j}$.

Definition 3: For an NoC architecture graph $\top' = G(N, L)$, a deterministic routing algorithm, $\Re : L \to P$, maps $l_{i,j}$ to one candidate routing path $p_{i,j} \in P_{i,j}$.

Definition 4: For mapping a task graph $\top = G(V, E)$, onto an NoC architecture graph $\top' = G(N, L)$, a function $map : T \to N$ is given, such that, $map(v_i) = n_j$, $\forall v_i \in V, \exists n_j \in N$.

5.2.2.2 Parameters and Variables

Based on the former definitions, the parameters and variables listed below are used for the problem formulation:

- An application task $= v$ belongs to a task graph $= V$.
- The source task and the destination task of a communication flow $= v_{src}$ and v_{dst}, respectively.
- Communication volume of a task pair $= comm(e_{i,j})$.
- Required bandwidth of a task pair $= b(e_{i,j})$.
- A network node $= n$ belongs to a node graph $= N$.
- The source node and the destination node of a communication routing path $= n_{src}$ and n_{dst}, respectively.
- Communication volume of a node pair $= comm(e_{i,j})$.
- Link capacity of a node pair in the network $= bw(l_{i,j})$.
- Hop distance of a node pair n_i and $n_j = hop_{n_i,n_j}$.
- Energy cost of one bit data from node n_i to $n_j = e(l_{i,j})$.

- Timing cost of one bit data from node n_i to $n_j = t(l_{i,j})$.
- Number of tasks in an application $= size(V)$.
- Number of nodes in a network $= size(N)$.
- Coordinates of a node in a network: $n_i = (x_i, y_i)$.
- The column number of a Network $N = column(N)$.
- The row number of a Network $N = row(N)$.

5.2.2.3 Objective Function

The main objective of our mapping technique is to minimise the trade-off cost of the communication energy and the transmission delay, which contains two parts to be optimised as a whole. The objective function is constructed based on a Non-Linear Programming (NLP) formulation:

$$Minimise : z = \{Cost_E + Cost_T\} \tag{5.8}$$

By using above definitions, parameters and variables, the two polynomials in Equation 5.8 can be elaborated as follows:

$$\begin{cases} Cost_E = \sum\limits_{\forall e_{i,j} \in E} \sum\limits_{\forall l_{map(v_i),map(v_j)} \in L} \sum\limits_{\forall v_i, v_j \in V} comm(e_{i,j}) \cdot e(l_{map(v_i),map(v_j)}) \cdot hop_{map(v_i),map(v_j)} \\ Cost_T = \sum\limits_{\forall e_{i,j} \in E} \sum\limits_{\forall l_{map(v_i),map(v_j)} \in L} \sum\limits_{\forall v_i, v_j \in V} comm(e_{i,j}) \cdot t(l_{map(v_i),map(v_j)}) \cdot hop_{map(v_i),map(v_j)} \end{cases} \tag{5.9}$$

The two polynomials present the communication energy cost and transmission timing performance of the application task mappings, respectively. By summing the arithmetic products of communication volume between any task pair, the bit energy or timing between the corresponding node pairs to which each task pair has mapped times the hop count of those mapped node pairs. From **Definition 3**, the mappings between the task graph and the NoC architecture graph can be established:

if

$v_i, v_j \in V, \quad are\ mapped\ to \quad n_k, n_l \in N, \ respectively,$

then

$map(v_i) = n_k, \quad map(v_j) = n_l.$

Then, Equation 5.9 is represented as shown in Equation 5.10:

$$\begin{cases} Cost_E = \sum\limits_{\forall e_{i,j} \in E} \sum\limits_{\forall l_{k,l} \in L} \sum\limits_{\forall n_k, n_l \in N} comm(e_{i,j}) \cdot e(l_{k,l}) \cdot hop_{n_k,n_l} \\ Cost_T = \sum\limits_{\forall e_{i,j} \in E} \sum\limits_{\forall l_{k,l} \in L} \sum\limits_{\forall n_k, n_l \in N} comm(e_{i,j}) \cdot t(l_{k,l}) \cdot hop_{n_k,n_l} \end{cases} \tag{5.10}$$

Since the dimensions of communication energy and transmission time are different, their results cannot be directly added to describe the minimal sum cost. Moreover, simply adding the two performance results may cause different weighted proportions in analysing the result optimality due to different orders of magnitude held by the result values. Hence, to seek the optimal sum trade-off with equivalent significance, both performance metrics are normalised to make their values convergent, eliminating the impact of dimensions and magnitudes.

Several normalisation methods are commonly used in many engineering fields to remove dimensions of different experimental results. The most popular one is *rescaling*, establishing relations between the maximal and minimal answers to rescale all answers at range $(0, 1)$. The others include *logarithmic function* and *inverse tangent function*. We do not use the rescaling method due to the difficulty of finding an accurate maximal answer to our mapping problem. The accurate worst case is hardly achieved unless all possible solutions are experimented their ultimate results and mutually compared. But this exhaustive searching is too expensive to be implemented in our case. Therefore, for efficient execution, the maximal solution to our problem, which is also the worst mapping, has been banned in early processing steps, which prohibits the use of the rescaling method.

For the other two methods, a simple experiment to test their execution speeds for solving our mapping problem shows that the logarithmic normalisation costs slightly less execution time. Thus, it is adopted as the normalisation method in our formulated problems for eliminating large magnitude differences of performance results. Moreover, to ensure the logarithmic results of both energy and time to be located at the same range for eliminating the dimensionality, the value units need to be clarified. Specifically in our case, the units of communication energy and transmission time are set as nJ and ns. The normalised objective function is clearly defined in Equation 5.11:

$$\begin{aligned} &Minimise : z = \{Norm\_Cost_E + Norm\_Cost_T\} \\ &\textbf{where :} \\ &Norm\_Cost_E = \log_{10}(Cost_E \ (nJ)) \\ &Norm\_Cost_T = \log_{10}(Cost_T \ (ns)) \end{aligned} \tag{5.11}$$

Thus, with the polynomials and pre-defined variables given in Equation 5.10, the modified main objective function of the application mapping problem can be expressed:

$$
\begin{aligned}
Minimise: z = \Bigg\{ &\log_{10}\Big(\sum_{\forall e_{i,j} \in E} \sum_{\forall l_{k,l} \in L} \sum_{\forall n_k, n_l \in N} comm(e_{i,j}) \cdot e(l_{k,l}) \cdot hop_{n_k,n_l} \Big) \\
&+ \log_{10}\Big(\sum_{\forall e_{i,j} \in E} \sum_{\forall l_{k,l} \in L} \sum_{\forall n_k, n_l \in N} comm(e_{i,j}) \cdot t(l_{k,l}) \cdot hop_{n_k,n_l} \Big) \Bigg\}
\end{aligned}
\tag{5.12}
$$

For the polynomial hop_{n_k,n_l} in Equation 5.12, the aforementioned mapping algorithms like [119] and [15] usually calculate hop distance values by denoting a new parameter $(d^k, k = 1, 2, ..., |E|)$ to present a single communication commodity between a node pair, and summing the values of all the commodities $(value(d^k))$ which are available along the directional routing path for a specific communication flow:

$$
hopcount = \sum_{k=1}^{|E|} value(d^k) \cdot p(n_{src}, n_{dst}), \ \forall p_{i,j} \in P_{i,j}
$$

This method needs an efficient and specific routing scheme adapted to certain NoC topologies, such that the calculative results can be optimised.

As we intend to expand the usability of the problem formulation to more topologies other than Mesh-based ones, a new method that calculates the hop count using a coordinate system has been developed in this chapter to express the interrelations of task-mapped nodes. Previous works often describe the node positions and their interrelations by summing along directional routing paths, which requires the characteristics of specific NoC topologies. This limits the use of such description methods for other NoC architectures. Our method elaborates the interrelations of network nodes by using the absolute coordinate values of node positions, which forms non-linear expressions. This NLP-based method can be used in other NoC topologies since no specific network and routing characteristics are involved, which improves the universality.

For those tile-based NoC architectures whose node interconnections can be depicted by the binary coordinate system, like Mesh and Torus for example, their coordinating interrelations can be expressed below:

$$\begin{aligned}
&\textbf{if}\\
&\quad \textit{the coordinates of a pair of task mapped NoC nodes are}:\\
&\quad n_i = (x_i, y_i), \quad n_j = (x_j, y_j),\\
&\textbf{then}\\
&\quad hop_{n_i,n_j} = |y_j - y_i| + |x_j - x_i|
\end{aligned} \tag{5.13}$$

The method to elaborate node interrelations in other tile-based NoCs is similar. If more complex architectures are needed to formulate the mapping problem by our method, more complex coordinate systems (for example, ternary system) can be simply used. Taking such non-linear assumptions into consideration, the expression of the objective function in our mapping problems has finally be derived in Equation 5.14:

$$\begin{aligned}
Minimise: z = \Bigg\{ &\log_{10}\Big(\sum_{\forall e_{i,j}\in E}\sum_{\forall l_{k,l}\in L}\sum_{\forall n_k,n_l\in N} comm(e_{i,j})\cdot e(l_{k,l})\cdot(|y_l-y_k|+|x_l-x_k|)\Big)\\
&+\log_{10}\Big(\sum_{\forall e_{i,j}\in E}\sum_{\forall l_{k,l}\in L}\sum_{\forall n_k,n_l\in N} comm(e_{i,j})\cdot t(l_{k,l})\cdot(|y_l-y_k|+|x_l-x_k|)\Big)\Bigg\}
\end{aligned} \tag{5.14}$$

5.2.2.4 Constraints

Besides the objective function, several constraints below are also significant to acquire a proper and feasible formulation to our proposed application mapping problem.

1. *Graph-size constraint:*

$$\begin{aligned}
&\textbf{given}\\
&\quad \textit{a task graph } \top = G(V,E) \textit{ and a NoC architecture graph } \top' = G(N,L),\\
&\textbf{satisfy}\\
&\quad size(\top) \le size(\top')
\end{aligned} \tag{5.15}$$

Equation 5.15 indicates the proper size of NoC architectures that have sufficient size of nodes and links onto which the tasks of candidate application with certain communications should be carefully mapped. It ensures that there are no tasks or communications unable to be mapped onto a specific network. By considering this constraint, the least available size of an NoC architecture can be determined.

2. *Task-node corresponding constraint:*

$$\forall v_i \in V, \quad map(v_i) = n_i \in N \tag{5.16}$$

Equation 5.16 enables any single application task belonging to a task graph to be mapped onto a node belonging to the same specific NoC architecture graph. It ensures all the tasks in the task graph are mapped onto the candidate network nodes, which is essential to construct a complete communication flow in networks.

3. *One-to-one mapping constraint:*

$$\forall v_i \neq v_j \in V, \quad map(v_i) \neq map(v_j), \quad i.e. \quad n_i \neq n_j \in N \tag{5.17}$$

This equation ensures a one-to-one mapping in the proposed problem formulation, such that each individual task in a task graph should be mapped onto an independent node of the corresponding NoC network. Since each network node only has one single IP core containing one individual task in our proposed problem, the feasible task routing in the network is guaranteed to optimise its desired performance metrics. Considering the non-linear coordinate expressions of hop counts used in our problem (Equation 5.13), the following coordinate constraint of task-mapped nodes is derived based upon the given constraints in Equation 5.15 and Equation 5.17:

$$\begin{aligned} &\because size(\top) \leq size(\top') \ and \ \forall n_i, n_j \in N \Rightarrow n_i = (x_i, y_i), \ n_j = (x_j, y_j); \\ &\therefore \forall (x_i, y_i) \neq (x_j, y_j), \quad if \ n_i \neq n_j \in N. \end{aligned} \tag{5.18}$$

4. *Bandwidth constraint:*

$$\forall l_{k,l} \in L, \ bw(l_{k,l}) \geq \sum_{\forall e_{i,j} \in E} b(e_{i,j}) \cdot \sum_{\forall n_k, n_l \in N} l_{k,l}^{i,j} \tag{5.19}$$

where $l_{k,l}^{i,j}$ is defined:

$$l_{k,l}^{i,j} = \begin{cases} 1, \ if \ map(v_i) = n_k \ and \ map(v_j) = n_l \\ 0, \ otherwise \end{cases}$$

The bandwidth constraint specified in Equation 5.19 is another critical factor for formulating proper and feasible communication flows of application tasks on NoC architectures. The available bandwidth of NoC channel links between a task-mapped node pair should always be larger than or equal to the communication volume required by the mapped task pair, which avoids potential communication overflow at this task pair and network failure caused by heavy traffic workloads.

More precisely, this bandwidth constraint ensures that the communication traffic flow of each task pair on the mapped network does not exceed the available channel link bandwidth, satisfying the bandwidth requirements between each mapped node pair in the network. The bandwidth performance parameter also implicitly impacts on the data packet transmission time, which is one of the major objectives for performance optimisation.

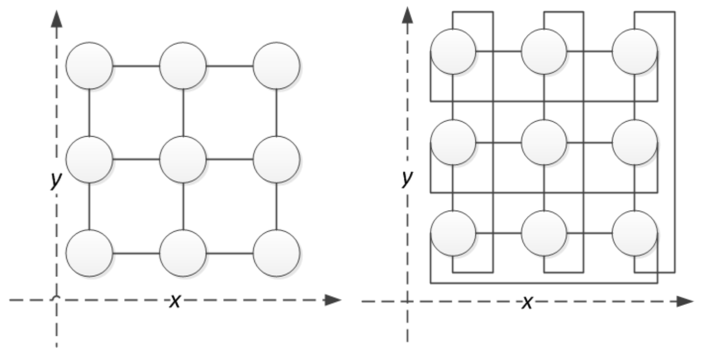
Heretofore, the proposed problem for application task mapping has been formulated thoroughly for further performance optimisation, associated with a set of constraints to ensure the feasible and proper elaboration. The completed problem formulation is specified in Equation 5.20:

$$
\begin{aligned}
Min : \Big\{ & \log_{10}\Big(\sum_{\forall e_{i,j} \in E} \sum_{\forall l_{k,l} \in L} \sum_{\forall n_k, n_l \in N} comm(e_{i,j}) \cdot e(l_{k,l}) \cdot (|y_l - y_k| + |x_l - x_k|)\Big) \\
& + \log_{10}\Big(\sum_{\forall e_{i,j} \in E} \sum_{\forall l_{k,l} \in L} \sum_{\forall n_k, n_l \in N} comm(e_{i,j}) \cdot t(l_{k,l}) \cdot (|y_l - y_k| + |x_l - x_k|)\Big) \Big\} \\
\textbf{subject to:} & \\
& size(\top) \leq size(\top') \\
& \forall v_i \in V, \quad map(v_i) = n_i \in N \\
& \forall v_i \neq v_j \in V, \quad map(v_i) \neq map(v_j), \quad i.e. \quad n_i \neq n_j \in N \\
& \forall (x_i, y_i) \neq (x_j, y_j), \quad if \ n_i \neq n_j \in N \\
& \forall l_{k,l} \in L, \ bw(l_{k,l}) \geq \sum_{\forall e_{i,j} \in E} b(e_{i,j}) \cdot \sum_{\forall n_k, n_l \in N} l_{k,l}^{i,j}
\end{aligned}
\tag{5.20}
$$

5.2.2.5 Extension of Mapping Problem onto More Tile-based NoCs

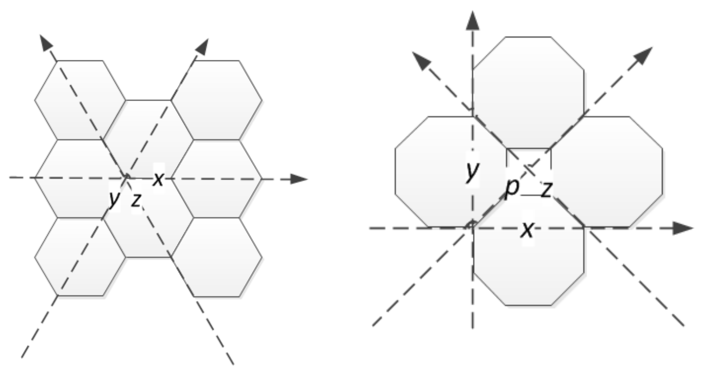
By solving the minimisation problem formulated in Equation 5.20, an optimal mapping of application tasks onto specific network nodes in a mesh architecture can be achieved. The optimised sum trade-off of the normalised communication energy and the normalised transmission time under constrained network bandwidth is also generated. An (x, y) binary set of coordinates is used in the mesh topology to program the interrelations of task-mapped nodes, which is extensible to elaborate node correlations of other tile-based NoC topologies. In other words, any NoC architecture whose node connections are capable of being illustrated in a non-linear way is suitable for the proposed non-linear programming based problem formulation.

Specifically, such a way to elaborate the node correlations by using a coordinate system is feasible for many popular regular and/or tile-based NoC topologies like Torus [36], Hexagon [180] and Octagon [175] [181]. The only difference in the coordinate usage on these topologies is the different coordinate systems needed to express their node system positions, which may cause different complexity on the elaboration of node interrelations. The Mesh (Figure 5.3(a)) and Torus (Figure 5.3(b)) topologies may need a binary set of coordinates (x, y) (as shown in Equation 5.13) to present their node correlations and mathematically formulate the relevant mapping problems.



(a) Coordinate System for Mesh Topology (b) Coordinate System for Torus Topology

Figure 5.3: Coordinate Systems for Mesh and Torus Topologies



(a) Coordinate System for Hexagonal Topology (b) Coordinate System for Octagonal Topology

Figure 5.4: Coordinate Systems for Hexagon and Octagon Topologies

However, such a set of coordinates may not be enough to thoroughly elaborate node interrelations in Hexagon (Figure 5.4(a)) and Octagon (Figure 5.4(b)) topologies. More complicated coordinate systems, ternary (x, y, z) and quaternary (x, y, z, p) sets of coordinates specifically, are needed to program their interrelating positions with non-linear expressions. To extend the proposed NLP formulation for Hexagon and Octagon architectures, the mathematical polynomial in Equation 5.20 should be changed in

Equation 5.21. Other equations need to modify their relevant polynomials accordingly.

$$\begin{aligned}
&\textbf{if} \\
&\quad \textit{the coordinates of a task mapped node pair are}: \\
&\quad n_i = (x_i, y_i, z_i), \quad n_j = (x_j, y_j, z_i) \ \textit{in Hexagon}, \\
&\quad n_i = (x_i, y_i, z_i, p_i), \quad n_j = (x_j, y_j, z_i, p_i) \ \textit{in Octagon}. \\
&\textbf{then} \\
&\quad hop_{n_i,n_j} = |y_j - y_i| + |x_j - x_i| + |z_j - z_i| \ \textit{in Hexagon}, \\
&\quad hop_{n_i,n_j} = |y_j - y_i| + |x_j - x_i| + |z_j - z_i| + |p_j - p_i| \ \textit{in Octagon}.
\end{aligned} \tag{5.21}$$

The main advantage of this NLP-based elaboration method is its extensibility for expressing many NoC topologies other than Mesh in the same way, reducing the design cycles for formulating various mapping problems on various NoC topologies. The cost of this method is the different complexity in constructing NLP-based node interrelations, which may lower the efficiency of solving the corresponding mapping problems. In this thesis, in order to efficiently verify the functions and performance, the proposed method is only used to elaborated the mesh topology in experiments.

5.2.3 Efficient Mapping

In this section, a modified Branch and Bound (BB) algorithm is developed to search optimal mappings of our formulated problems with improved execution efficiency. The methodology of generating early-stage accurate performance prediction is also given.

5.2.3.1 Modified Branch and Bound Algorithm

The NLP-based formulated mapping problems can be solved by linear programming techniques for exact optimal mappings. However, the execution time cost of such techniques is expensive since estimating the uncertainty of nonlinear constraints and the objective function is difficult. The overhead of LP-based execution grows exponentially with the increased complexity of formulated problems, giving rise to an intolerably long time for solving target problems.

Previous popular mapping algorithms also cannot well tackle nonlinear optimisation due to the hard convergence of uncertain nonlinear constraints, leading to low execution efficiency for searching optimal mapping solutions and less-accurate search results. For example, evolutionary techniques like GA, PSO, ACO and their hybrid work cannot properly transform nonlinear constraints for existing algorithms to be refined. This impairs the accuracy of mapping results. Constructive algorithms with iterative improvement like NMAP, Onyx and Citrine may give near-optimal results, but at the cost of long execution time and a large number of iterations. Constructive algorithms without iterative improvement like PMAP, BMAP and CMAP may give less-accurate optimisation results when used for nonlinear optimisation. It is because such algorithms can hardly give reliable pre-configuration and construction from partial solutions of nonlinear problems. Hence, the hope for resolving NLP problems with good balance between performance and efficiency rests on deterministic mapping algorithms. The Branch and Bound algorithm is one such algorithm that is compatible with constrained nonlinear optimisation problems and can produce global optimality due to its tree searching mechanism. If its searching efficiency can be improved, the desirable mapping results and performance are achievable.

For such reasons, an original BB algorithm is modified in our method to accelerate the execution speed, while ensuring the optimality of searching results. The proposed BB algorithm is derived from previous work in [118] and [116]. It has two resolution steps. In the first step, a set of candidate task mappings are computed using tree searching. At each level of the search tree, the task mapping with the best performance is found by alternative branch and bound operations. Once a mapping candidate with the best cost is calculated at each level, it is further expanded along a deadlock-free XY routing path until all the sorted tasks are mapped. In the second step, operations in the first step are iteratively implemented until all the candidate mappings have been derived. The normalised sum cost of communication energy and transmission time are evaluated and the one with minimal value is chosen as the final optimal solution.

Specifically, we initially prioritise tasks depending on their maximum required bidirectional communication volumes (v_i for $\sum_{\forall i \neq j} \{comm(e_{i,j}) + comm(e_{j,i})\}$). Then the remaining unmapped tasks are placed onto the network nodes with maximum communications to the task or with the maximum number of neighbours. The searching mechanism in this step alternately runs the branch and bound operations to compute the optimal mapping result at each tree level:

- In the branch operation, firstly choose a mapped but unexpanded node from the search tree as the root node (The root node indicates the mapped node used as the base for expanding remaining unmapped nodes to unmapped tasks). Then the unmapped task that has maximum communications with the already mapped task on the chosen node is successively mapped to a set of remaining unoccupied nodes in the network to generate a set of corresponding child nodes at the new level.

- In the bound operation, the newly generated child nodes at the same level are inspected to see if they can generate descendant nodes with optimal performance. The set of nodes are greedily searched (that is, to find the locally optimal choice at each step) to compute the normalised sum cost of communication energy and timing cost. Subsequently, the child node with legal minimum cost at this level is specified as the optimum after cost comparisons. Nodes with higher calculated cost are trimmed from further expansion. The legal condition indicates that the needed communication of mapped tasks is no more than an available link bandwidth along the routing path.

Figure 5.5 shows an example search tree structure used in our approach. The remaining unmapped tasks are sorted initially. Denote one unexpanded mapped node as the root node, then search its neighbouring nodes in the X-direction or Y-direction for mapping the most correlated tasks. This branch operation generates child nodes at a new level, and the normalised sum cost is computed in the bound operation to find the minimum cost mapping (or maybe mappings) of this level. They are deemed promising mappings for further expansions to the next levels. Other child nodes with higher costs will be trimmed. After all the sorted tasks are mapped, the mappings at the lowest level form the candidate mapping set. The optimal mapping will be produced from the set by calculating and comparing their performance values.

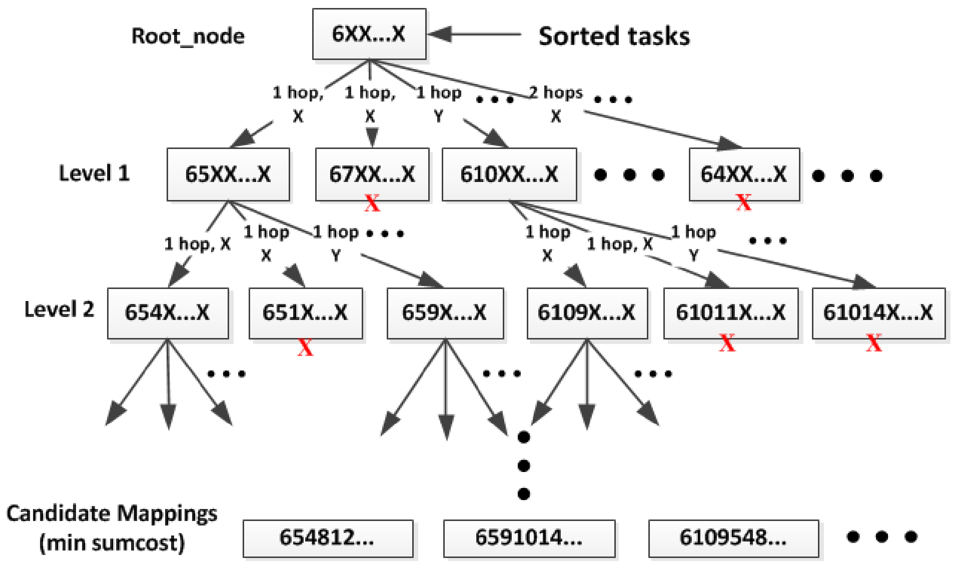


Figure 5.5: Search Tree Example

The task sorting and early node mapping mechanism in our algorithm could help detect and trim unpromising nodes earlier and reduce the number of expanded nodes, which has a larger impact on the improvement of energy and timing costs. The generation of a new child node (n_i) from unoccupied node set (N') for the unmapped task is improved. The node search starts from adjacent neighbours of the root node (n_{root}). For further increasing the execution efficiency of our algorithm, the searching is set for candidate child nodes whose hop distance to root node is no more than half of the network diagonal hop distance, instead of searching the unmapped nodes with all distances. For example, such a searching modification in a mesh is given in Equation 5.22, where $column(N)$

and $row(N)$ present the column number and row number of the network, respectively.

$$\begin{aligned} &\forall n_i \in unmapped\ node\ set\ N' \in the\ Network\ N, \\ &1 \leq hop_{n_i,n_{root}} \leq \frac{1}{2}\left[(column(N)-1)+(row(N)-1)\right] \end{aligned} \tag{5.22}$$

The reason that the network diagonal hop distance is set as a searching boundary is because we always start the sorted task mapping from the centre network nodes whose hop distance to the corner and border nodes are normally no more than the network diagonal hop distance even in the worst case. So although a rare chance exists that an optimal mapping between a task pair is longer than this distance, in most cases there is no need to search other candidates in the set that has more hops than this. Moreover, placing one task onto a node with a longer hop distance than the diagonal one means a long routing interval between the mapped task pair for data to traverse across more than half a network. This may make the energy and time cost of such routings mostly impossible to be optimised to the minimum since both the performance metrics are closely correlated and severely affected by hop distance.

Based on Figure 5.5 and Equation 5.22, an example of generating the specific candidate mapping 654812... from the leftmost trees in Figure 5.5 is given to illustrate our mapping algorithm. The generation of other candidate mappings may also be mentioned.

- Firstly, a specific application is split into a number of tasks with communication volumes to each other. Communication volumes are the data bits transmitted from and to one given task. Among all these unsorted tasks, No.6 task has the maximal communication volumes that is chosen to be mapped onto the central node of the network. This mapped node is considered as the root node for expanding other unmapped nodes to the remaining unmapped tasks.

- At the next level (Level 1) in Figure 5.5, tasks that have the maximal communication volumes to the No.6 task are mapped onto nodes that are adjacent to the root node. The task expansion rule is based on the hops between the root node and the candidate unmapped nodes, starting from one hop in the X direction to one hop in the Y direction, two hops in the X direction, two hops in the Y direction and so on. It is noted that more than one task may be valid for expansion. Particularly in this figure, No.5 task and No.10 task both have the maximal communication volumes to the No.6 task while No.7 task does not. Hence in this level, the branch operation generates all the available task mappings. The bounding operation, depending on whether the communication or hop-distance constraints are matched, reserves the No.5 and No.10 tasks but trims the No.7 task. The remaining task mappings in this level (node mappings of task pairs $6-5$ and $6-10$) are kept for subsequent levels of node expansion.

- Focusing on the Level 2 of leftmost trees in Figure 5.5 (the node mapping of task pair $6-5$), the node mapped with task 5 is the new root node for further expansion. By inspecting the remaining unmapped tasks, No.4 and No.9 tasks have the maximal communication volumes to the No.5 task. Thus these tasks are mapped onto adjacent nodes (from 1 hop along the X or Y directions) to the node mapped with the No.5 task, constructing the valid node mappings of tasks $6-5-4$ and $6-5-9$ for subsequent levels of node expansion. Other task mappings generated in the branch operation like tasks $6-5-1$, $6-10-11$ and $6-10-14$ in Figure 5.5 are eliminated in the bounding operation due to the unmatched constraints. It is observable from the figure that the node mapping of tasks $6-10-9$ also satisfies the bounding conditions, which is also kept at this level.

- At Level 3 and higher levels, the node mapping of tasks $6-5-4$ is kept expanding with the branch and bound operations run at each level until all the tasks are mapped onto network nodes. The final candidate mapping of tasks $6-5-4-8-12-...$ at the leftmost side in Figure 5.5 is therefore produced. Meanwhile, other node mappings that satisfy the bounding constraints at each level, such as tasks $6-5-9-10-14-...$ and $6-10-9-5-4-8-...$ in the figure, are also kept and stored as the final candidate mappings. These remaining mappings are all possible optimal ones for the specific application. Hence they are all stored in the candidate pool. Once the pool of final candidate mappings is completely produced, the whole algorithm stops. At this step, the normalised cost of energy and time is calculated based on each candidate mapping to carefully compare their results. The mapping with the minimal results is chosen as the optimal mapping of the application for further operations.

As the cost of our modification, the proposed algorithm has a tiny optimality drop due to a negligible chance of optimal mappings found beyond the distance limit. But deadlock freedom is guaranteed by our algorithm with reduced search boundaries for candidate mappings, resulting in fewer unpromising branches involved at each level needing to be trimmed. Hence, the searching efficiency is substantially improved by spending less time to search among a reduced candidate set. The pseudo code of our modified BB mapping algorithm is given in Figure 5.6.

To summarise, two major modifications are given in our BB mapping algorithm for the NLP problem resolving, comparing to other BB algorithms [187]:

- The first major modification is how the unmapped network nodes expand. In our algorithm, once a task is mapped to a network node, the adjacent nodes to this mapped node are explored first for the unmapped tasks that have the closest communications to the mapped task. If all adjacent nodes (that is, nodes that have

```
$Sort\ Tasks\ upon\ communication\ volume;$
$root\_node = new\ Node(NULL)$
$sumcost = 0;\ hop_{n_i,n_j} = 0;\ D = diagonal\ hop\ distance;$
$Max\_comm = Task.maxcomm(v_i);$
$for\ each\ n_i \in N'$
$\{$   $if\ (n_i.neighbour = Maxcomm.neighbour):$
        $root\_node = n_i;$
    $else$
        $root\_node = Node.Maxneighbour(n_i);$
$\}$
$Map.insert(root\_node);$
$While(!Task.empty())$
$\{$   $for\ each\ n_i \in N'$
        $\{if\ (hop_{n_i,n_j} = 1\ available)$
        $\{$   $search\ along\ X - Y\ path;$
            $cur\_node = Task.Maxcomm(Map.Task);$
        $\}$
        $else\ if\ (It\ hasn't\ found\ unmapped\ node\ \ hop_{n_i,root} \leq D)$
        $\{$   $hop_{n_i,root}++;\ search\ along\ X - Y\ path;$
            $cur\_node = Task.Maxcomm(Map.Task);$
        $\}$
        $generate\ child\ node\ n_{New};$
        $n_i.sumcost = sumcost + sumcost.\left[cur\_node \times hop_{Map.Task,n_i}\right];$
        $sumcostlevel_i = min(n_i.sumcost);$
        $if(n_{New}.sumcost = sumcostlevel_i)$
          $Map.insert(n_{New})$
$\}$
$OptimalMap = Map.FinalSet$
```

Figure 5.6: Pseudo Code of Proposed Mapping Algorithm

only 1 hop from the mapped node) are occupied, the nodes that have two hops to the mapped one will be explored. In a word, our algorithm expands unmapped nodes based on the communication volumes (data bits) linked to the mapped task, instead of the total communication volumes one unmapped task holds, as in other BB algorithms.

This optimisation leads to a compact deployment of network nodes mapped to tasks with close communications, leading to a fast branching of promising/unpromising mappings to raise the execution efficiency.

- The second major modification is adding a distance constraint to the node exploration. In our mapping algorithm, the unmapped network nodes that have a specific hop distance (no longer than half of the network diagonal hop distance) to the mapped node are considered for expanding to the unmapped tasks. Since the hop distance is highly related to the communication energy and transmission time tasks may consume, which are the major objectives we would like to optimise, this modification constrains the hops between task pairs which reduces the total hops for an application as much as possible.

  This optimisation gives a smaller candidate pool for optimal mappings because a large number of mappings (which dissatisfy this distance constraint) are already trimmed away as unpromising results in the bounding operation at different levels. The smaller candidate pool can accelerate the execution speed while reducing the computational and storage overheads.

5.2.3.2 Performance Prediction

Optimised mappings to specific applications are the common outcomes of mapping techniques. The result optimality and performance metrics of produced mappings are diversely analysed in previous work based on various design aims. In our mapping method, optimisation of energy and time performance on various NoC architectures is considered, likewise a desirable balance between mapping performance and execution productivity. So far, the NLP based formulation technique and a modified BB algorithm are proposed to ensure extensibility and efficiency. Additionally, a method for calculating the verified mapping performance for reliable early prediction is further developed.

Due to the contradictory metrics of performance and efficiency, trading efficiency for performance improvement such as in a multi-stage result refinement and a large number of iterations in many previous work is not feasible in our method. Instead, a balanced trade off between mapping result precision and execution efficiency is established by integrating more accurate models in the problem formulation stage and calculating the performance results immediately after the optimal mappings are generated. In a word, we ensure the accuracy without sacrificing efficiency by 'precisely' predicting the performance from formulated models rather than 'iteratively refining' the results from mappings.

Specifically, the exact values of communication energy and transmission time are calculated directly after a mapping is produced. The accurate calculation uses baseline references of our energy and timing models derived from the NIRGAM simulator. According to the derived models in Equation 5.6 and Equation 5.7, values of variables E_A, E_B, T_A and T_B are retrieved by sending one bit of data into NIRGAM mesh networks and recording the simulation results. Such baseline values are substituted back into our performance models to underpin the calculation of accurate mapping results.

The accuracy of the calculated performance is verified by importing the produced mappings in NIRGAM and comparing with the simulation results. The whole process of our mapping technique, as shown in Figure 5.7, combines the given NLP based formulation and a modified BB mapping algorithm for mapping generation, then directly calculating results upon accurate performance models for performance prediction. Based on the flowchart, experiments are designed to validate our methodology.

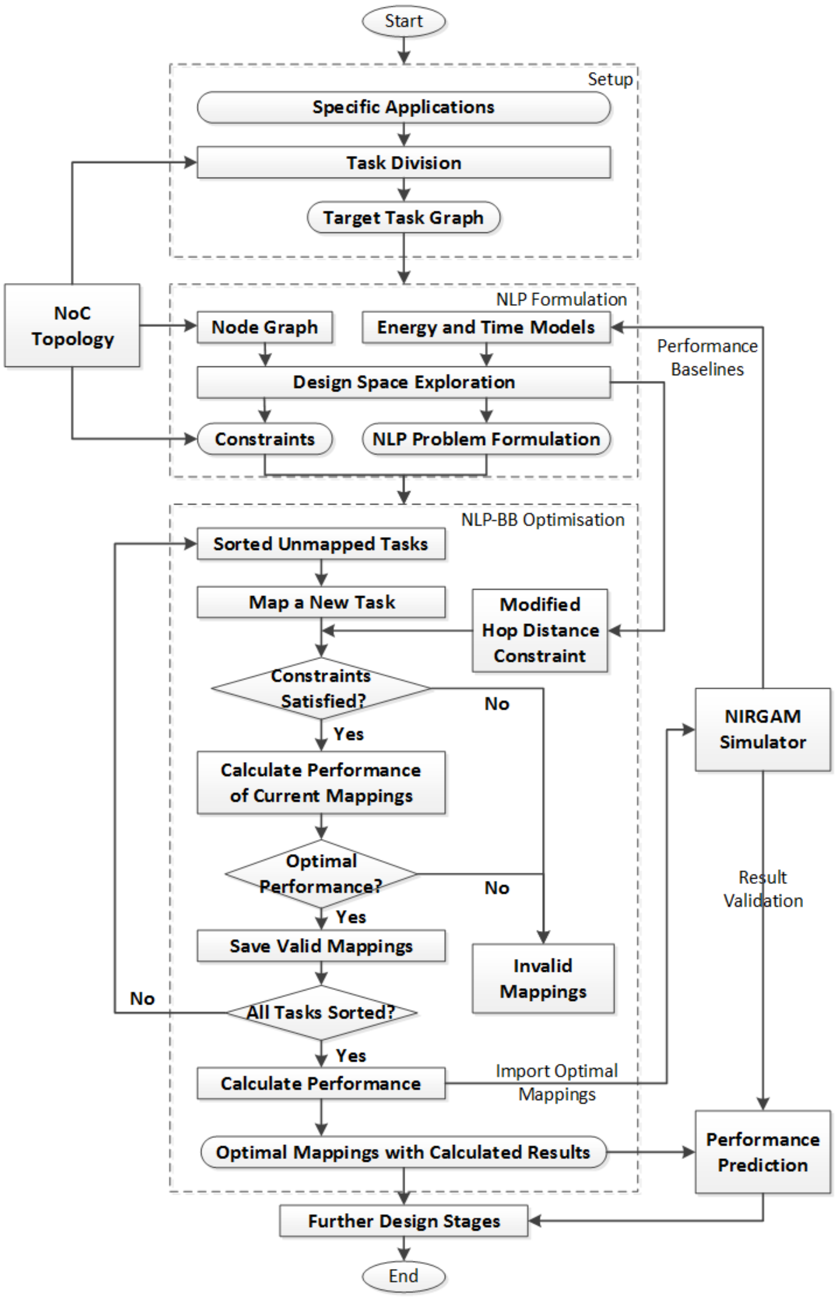


Figure 5.7: Proposed Method of Performance Prediction

5.2.3.3 Statement

To clarify how the differences between our proposed BB algorithm and previous work using a revised BB algorithm like in [187] are made, several statements are made in the following. Firstly, the initial settings for the branching phase are different. The initial setting of the referenced work is different from ours. The authors in [187] set the candidate tasks based on a descending order of communication volumes held by each task, while our algorithm orders the tasks based on the communication volumes held with the mapped task. So we map the tasks by concentrating more on the interrelations of communications between task pairs, instead of concentrating on the total communications one particular task has. The mappings applying our task order to start branching will converge faster on the potential optimal mappings. Moreover, by considering both the communication demands between task pairs and self communication volumes in our algorithm, more unpromising mappings from all possible solutions can be identified at earlier stages (higher levels), which leaves smaller candidate mapping sets level by level. This mechanism increases the efficiency of our algorithm.

Secondly, the algorithmic mechanisms at the bounding phase are different. The referenced work [187] maps one task to each of the available network nodes to calculate and compare the mapping cost, eliminating only those that have dominant cost and leaving a large amount of possible mappings in the candidate set. This keeps a large part of mappings reserved in the candidate set for seeking the optimal result, such that an user-defined threshold has to be made to restrain the set capacity. Extra non-dominant mappings are randomly eliminated from the solution set. Such operations may not only achieve nearly-optimal solutions (random elimination may cut off potential final optimal solutions), but also causes a computation and storage overhead. On the other hand, our modified BB algorithm has a smaller candidate set for the final optimum after branching. It searches those mappings that match all the conditions given in the formulated algorithm, ensuring the global optimality of solutions by searching all possible mappings instead of all available mappings (Note that these are two different kinds of mappings. For example, an available mapping may not be a possible mapping due to the failure of meeting some conditions and constraints like bandwidth or hop distance, while a possible mapping must be available). The smaller number of mappings needed for bounding in our algorithm increases the execution efficiency.

Thirdly, the objectives and performance factors are different. The main contribution of the referenced work in [187] is a dynamic mapping algorithm for optimising multi-objectives. Their work can achieve accurate pareto-optimal solutions based on real-time network conditions. Their focus is on dynamically-accurate, pareto-optimal mappings and efficient implementation with multiple objectives. Our algorithm focuses on providing statically-accurate, globally optimal mappings and efficient implementations with high execution speed. Different objectives and concerns result in different solution methods. The referenced work shown in [187] is not concerned much with execution speed, global optimality and computation overhead but our algorithm do. Similarly, our method is not concerned with the impact of dynamic network environment on mapping results but their work does. Because our method is expected to be extended for other NoC topologies, the NLP based formulation technique is developed. Because we want to achieve globally optimal, accurate mapping results with fast execution speed (for efficient performance prediction at early design stages), we have chosen the branch and bound algorithm and modified it to solve the formulated problems. A simulator is used in the referenced work to explore the mapping space for different algorithms. But in our method, a simulator is used to verify the algorithmic results. A simulator is used in [187] as a development tool and the simulator in our method is used as a verification tool. All these differences make our algorithm differ substantially from the referenced work.

Finally, compared to other state-of-the-art approaches, our method may also have differences. These are due to the different objectives and developing purposes. The reason is shown in our contributions: we developed a mapping method for extensible NoC topologies (why develop NLP formulation) to achieve globally optimal mappings within high-efficiency execution time (why develop modified BB algorithm for NLP formulation), providing accurate and reliable performance prediction (why develop simulation tool and prediction method) at the early NoC design stages. Our work is different from previous work in their reports.

5.3 Experimental Results

In this section, experiments are implemented to validate our NLP-BB mapping technique in the following respects:

- **Functional verification**: Our modified NLP-BB algorithm is expected to generate optimal mappings with the minimal trade-off cost of communication energy and transmission time. To verify this functionality, we compare the performance results of mappings produced by our algorithm with results of other algorithms.
- **Algorithm execution efficiency**: The efficiency of mapping algorithms to produce the optimal mappings is essential in our method for early-stage performance prediction. In subsequent experiments, the execution time costs of all algorithms to produce their optimal mappings are recorded and analysed.
- **Performance accuracy**: For accurate and reliable performance prediction, our method calculates the performance results of produced mappings. The result

accuracy is examined by comparing calculated results with simulation results generated by the same mapping in NIRGAM.

To meet these experimental objectives, a modern mapping algorithm, Interior Point Method (IPM) [188], and the original Branch and Bound (BB) algorithm [116] are introduced for mapping comparisons. The IPM is a class of algorithms designed for solving constrained nonlinear optimisation problems. Due to the nonlinear constraints, previous mapping algorithms like [108] are not compatible and their searching cannot converge rapidly in our formulated problems, leading to intolerably long execution time for mapping generation (A threshold of 1 day is set for manual termination of overtime execution). Thus, the IPM is introduced for fair comparisons.

The IPM searches from an initial point and traverses the interior of the feasible region until a nearest convergent point. Hence, it has fast convergence speed when optimising nonlinear problems. But such searching can easily be trapped locally. To tackle this in our experiments, we modified the IPM integrated in the MATLAB toolbox to let it run from each initial point (that is, starting from each network node), generate all local optimum and mutually compare them to produce a global optimum. Besides, the maximal iteration number of IPM is limited to $10,000$ for improving the searching efficiency.

Both synthetic applications and a real-world media application are given in the experiments and formulated using our NLP-based technique. All three algorithms are implemented to search the optimal mapping of each application. Since smaller logarithmic sum of energy and time can always be achieved by smaller values of both energy and time metrics (which are positive real numbers), the resulting optimality of all mapping algorithms is experimented via energy and time performance comparisons of the produced mappings. For the same application, both calculated and simulation mapping results generated by all algorithms are compared for functional verification. For the same mapping of each algorithm, the calculated results are compared with the simulation results for performance accuracy. The execution time cost of all algorithms to generate optimal mappings for every application is recorded for execution efficiency.

5.3.1 Synthetic Applications

5.3.1.1 Experimental Setup

Synthetic traffic benchmarks are extracted from the MCSL NoC benchmark suite [189] as applications for experiments on five different sizes of $2D$-mesh networks (network scales from 4×4 to 8×8). The task numbers and linked communications of extracted benchmarks form application traffic that cover all network nodes, which are increased along with the size of networks as shown in Table 5.1. For all applications, mapping problems are formulated using the proposed NLP-based techniques. Three mapping algorithms (modified IPM, original BB and our NLP-BB algorithm) are applied to search the optimal mapping of each problem and calculate energy and time results of each mapping. The execution time cost by each algorithm to generate the mapping is also recorded. All the produced mappings are imported to the NIRGAM for performance simulations. In each formulated problem, the IPM algorithm has been run 10 times and the mapping with the best calculated results is selected for comparison.

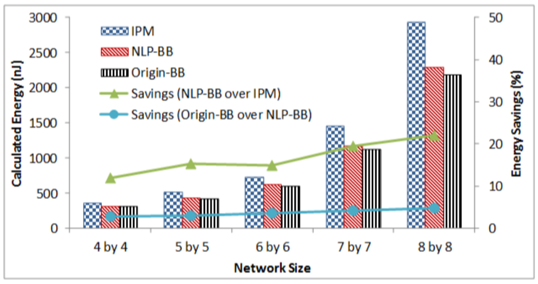
Table 5.1: Synthetic Benchmarks and Network Sizes

Applications	No.of Tasks	Network Size
C1	62	4×4
C2	324	5×5
C3	812	6×6
C4	1378	7×7
C5	1891	8×8

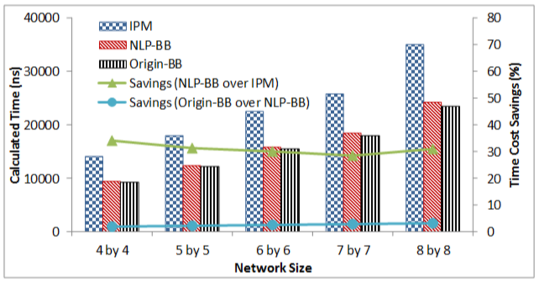
5.3.1.2 Functional Verification Analysis

Figure 5.8 and Figure 5.9 give performance comparisons of both calculated and simulation results of the optimal mappings produced by the three algorithms. From Figure 5.8(a) and Figure 5.8(b), the calculated energy and time cost of the mapping generated by the original BB algorithm is the lowest in each case, closely followed by our NLP-BB algorithm. The energy and time cost by IPM produced mappings are the highest in all case applications, which suggests the limited number of iterations set for IPM algorithm is inadequate to find better mappings. Our NLP-BB algorithm generates less-optimal mappings with on average 3.61% more energy and 2.47% more time cost than the original BB algorithm. This is due to the given hop-distance constraint causing a reduced solution set for exploration, leading to a slight performance drop of produced mappings. Compared to the IPM algorithm, our algorithm performs considerably better with on average 16.7% less energy and 30.87% less time cost.

In Figure 5.9(a) and Figure 5.9(b), mappings generated by all algorithms are imported to NIRGAM for simulating their energy and time cost, resulting in similar performance outcomes as the calculated results. Our NLP-BB algorithm performs on average 3.93% more energy and 1.9% more time than the original BB algorithm, and 16.23% less energy and 31.06% less time than the IPM algorithm. In both the calculated and simulation results of all case applications, our NLP-BB algorithm always generates mappings with very close performance to the mappings of the original BB algorithm, showing a competitive result optimality. Since the BB algorithm can produce global optimum due to its exhaustive searching mechanism, our algorithm proves that it can provide global optimality as well.

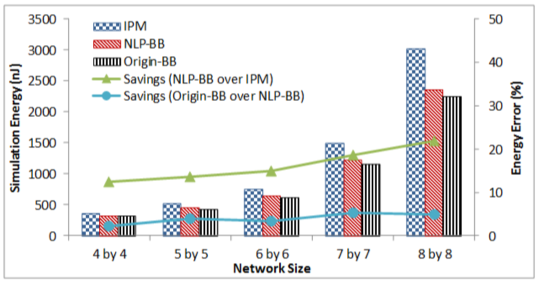


(a) Calculated Energy Comparison

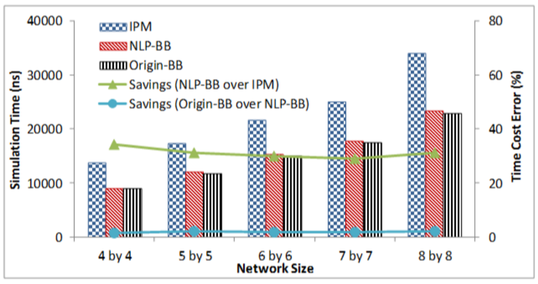


(b) Calculated Time Comparison

Figure 5.8: Comparisons of Calculated Results by Different Algorithms for Synthetic Applications



(a) Simulated Energy Comparison



(b) Simulated Time Comparison

Figure 5.9: Comparisons of Simulated Results by Different Algorithms for Synthetic Applications

5.3.1.3 Execution Efficiency Analysis

Algorithmic efficiency is an essential constraint in our method. Since our target is to efficiently solve the small-size to medium-size applications by our mapping algorithm, the execution time of mapping generation in all case applications could be recorded as it is a crucial factor to evaluate the algorithm quality. Table 5.2 presents the runtime of all three mapping algorithms for all case applications in seconds. The simulations are run on a desktop with 8 Intel Xeon 2.67GHz core processors and a 12GB memory. The execution time of mapping generation by the IPM algorithm is always the highest in experiments due to its costly iterative searching from different initial points. Comparing two BB algorithms, our NLP-BB algorithm consumes much less execution time since our searching solution set is smaller than the original BB algorithm. Moreover, the difference of execution time between our algorithm and other two grows substantially along with increased complexity of case applications. The superiority of our NLP-BB algorithm is clearly presented in Figure 5.10.

From the figure, two execution time ratios between our NLP-BB algorithm over the other two algorithms in all case applications are given, respectively. As the network size

Table 5.2: Runtime Records of Synthetic Applications Cost by Different Mapping Algorithms

Applications	IPM (s)	Original BB (s)	NLP-BB (s)
C1	324.7	19.6	6.7
C2	478.3	53.4	9.2
C3	948.1	123.5	13.5
C4	1982.6	321.8	22.8
C5	2889.2	583.6	29.3

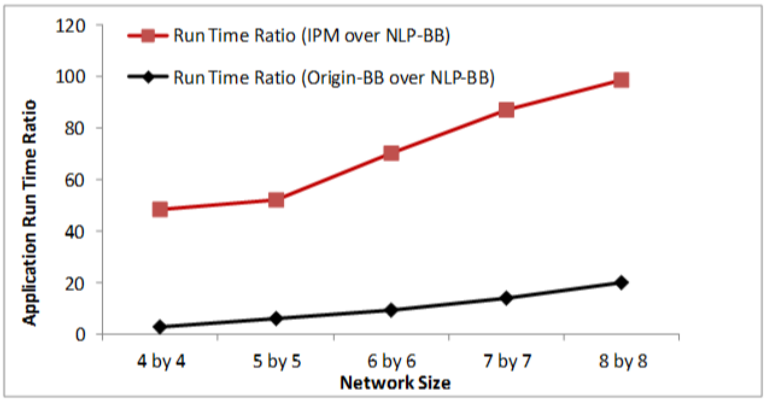


Figure 5.10: Run Time Comparisons

scales up, our NLP-BB algorithm runs from 48.46 times to 98.61 times faster than the IPM algorithm, and from 2.93 times to 19.92 times faster than original BB algorithm to produce the optimised mappings. On average, the original BB algorithm costs a 10.38-fold increase and the IPM algorithm has a 71.25-fold increase of execution time than our algorithm, indicating notably superior efficiency of our NLP-BB algorithm.

5.3.1.4 Performance Accuracy Analysis

The proposed mapping technique not only produces optimal mappings but also calculates their energy and time performance results as references for further design stages. To ensure reliable prediction, the resulting accuracy of the calculated mapping performance is analysed via comparisons with simulation results of the same mappings in NIRGAM. Figure 5.11 shows the performance error comparisons between calculated and simulated results in terms of energy and cost, covering every mapping generated by all three algorithms in all case applications. As observed from the figure, there is no big fluctuation among all performance metrics. All the kinds of errors are ranged from 2% to 4%. Hence, the calculated performance results by our method are validated to have little difference from simulation results, which displays accurate and reliable early-stage performance prediction.

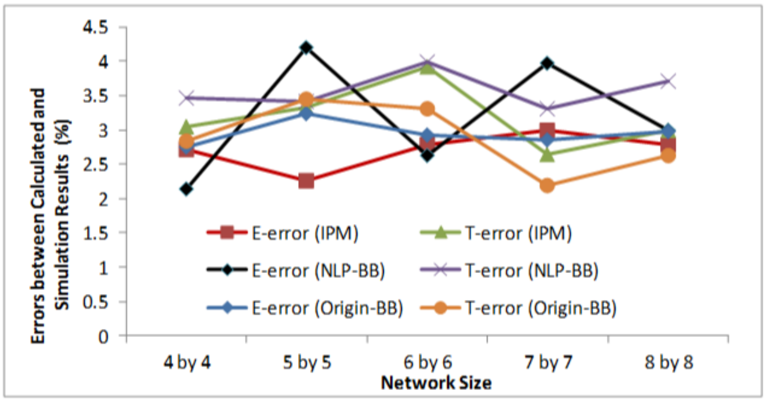


Figure 5.11: Error Comparisons Between Calculated and Simulated Results of Different Mapping Algorithms

Potential reasons for the errors are also investigated. We calculate the performance results based on baseline metrics E_A, E_B, T_A and T_B of Equation 5.6 and Equation 5.7. The values involved in referenced metrics may differ from real network simulations as each time fluctuant simulation results are generated in NIRGAM due to various routing delays at buffers or routers of intermediate nodes. The fixed values in models always produce the same calculated results, leading to inevitable experimental errors.

To summarise, in synthetic applications our NLP-BB algorithm runs 10 times faster than the original BB algorithm at the cost of only $(2 \sim 4)\%$ performance drops. Savings of both performance and execution time by our algorithm over IPM are substantial, which are $(16 \sim 30)\%$ and 71-fold, respectively.

5.3.2 Real-World Media Application

5.3.2.1 Experimental Setup

An MPEG-4 decoder is implemented to evaluate the potential of our method for real multimedia applications. It has been divided into 8 sections with different functions for successful simulations in NIRGAM. The input of the decoder is an encoded video bitstream of a QCIF-format MPEG-4 video clip with 1-second of content and 25fps frame rate. The output is YUV data of the reconstructed video clip. The MPEG-4 application has different communication volumes required by its sections, enabling our NLP-based mapping problem to be formulated.

The task graph of MPEG-4 decoding is given in Figure 5.12. Symbols between different sections in the figure represent total communication volumes (number of tasks) required by each section pair, which is given in Table 5.3. By using our method, the NLP-based mapping problem is formulated and solved by the IPM, original BB and our NLP-BB algorithms, respectively. To make the comparison fair, all mapping algorithms are running on a 4 × 4 2D-mesh network to explore the optimal mapping with minimal trade-off cost of energy and time.

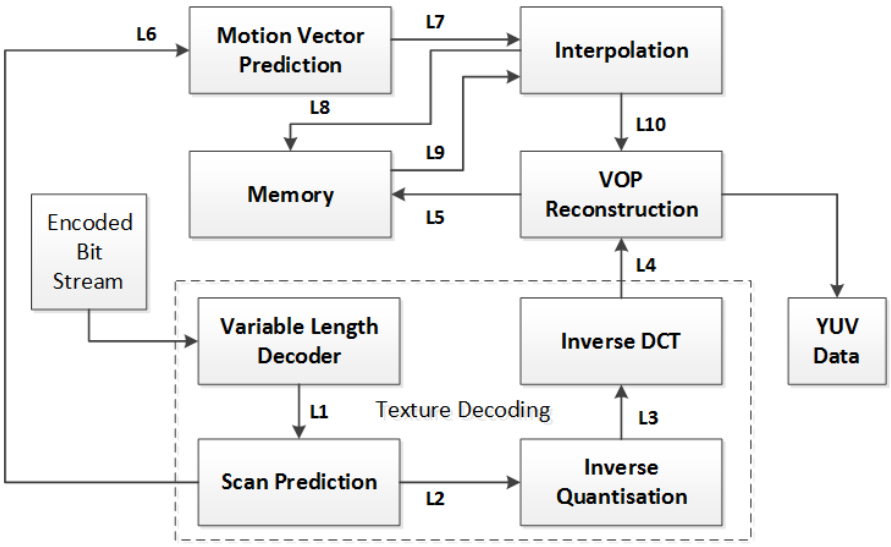


Figure 5.12: Task Graph of MPEG-4 Decoder

Table 5.3: Number of Tasks for Each Link

Communication Links	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10
Number of Tasks	833	2275	1724	1724	947	136	115	46	1111	933

Similar to synthetic applications, the generated optimal mappings of MPEG-4 decoding have been calculated the performance in terms of energy and time. The execution time of mapping generation by each algorithm is recorded and analysed. All the produced mappings are imported to the NIRGAM simulator for performance simulations. Both calculated and simulated results are listed in Table 5.4.

Table 5.4: Performance Results of Different Mapping Algorithms for MPEG-4 Decoding

Algorithms	IPM		Origin-BB		Our NLP-BB	
Metrics	Calculated	Simulation	Calculated	Simulation	Calculated	Simulation
Energy (μJ)	175.65	180.63	152.73	156.13	154.61	158.53
E-error (%)	2.76		2.18		2.47	
Time (μs)	5592.8	5549	5032.36	5007	5074.05	5043
T-error (%)	0.78		0.5		0.61	
Run Time (s)	9690.26	-	1187.13	-	94.33	-
Energy Improvement (%) of our NLP-BB			over IPM		11.98	12.24
			over Original BB		-1.22	-1.51
Time Improvement (%) of our NLP-BB			over IPM		9.28	9.19
			over Original BB		-0.82	-0.71
Run Time Improvement (X) of our NLP-BB			over IPM		102.73	-
			over Original BB		12.59	-

5.3.2.2 Functional Verification Analysis

From Table 5.4, the calculated performance results of the MPEG-4 decoding application produced by the three mapping algorithms are given. The energy and time cost of the mapping generated by our NLP-BB algorithm outperforms the IPM algorithm with 11.98% and 9.28%, respectively. The performance drops of energy and time metrics by our algorithm over the original-BB algorithm are minor, which are merely 1.22% and 0.82%, respectively.

The simulation performance results show similar situations as the calculated ones except that the values are slightly different. In NIRGAM simulations, the mapping produced by our NLP-BB algorithm has 12.24% and 9.19% savings in energy and time cost, respectively, compared to the IPM algorithm. The performance differences of our algorithm over the original BB algorithm are also tiny, costing 1.51% more energy and 0.71% more time in produced mappings.

5.3.2.3 Execution Efficiency Analysis

The execution speed of our NLP-BB algorithm in generating the optimal mapping of MPEG-4 decoding reveals great competitiveness, costing more than 102 (102.73 exactly) times faster than the speed of the IPM algorithm. The IPM spends 9690 seconds to produce a mapping that is less-optimal than ours whose time cost is only 94 seconds. This enormous efficiency gap is caused by the limited iterations for IPM algorithm to search such a complex application, suggesting a much longer time that the IPM needs to find the mapping as good as ours.

Moreover, the original BB algorithm still generates the mapping with slightly better performance than our NLP-BB algorithm, but at the cost of a vast increase (12.59-fold) in execution time. The original BB generates a more-optimal mapping than our algorithm due to a larger solution set available for searching. But its overhead of searching time for the better mapping evidently overwhelms its performance improvement, contrarily leading to a worse balance between result optimality and execution efficiency than our algorithm.

5.3.2.4 Performance Accuracy Analysis

Since our method calculates the performance results for early-stage prediction, the accuracy of calculated results are checked in synthetic applications by comparing their result errors with simulation performance, likewise the real media application. Through Table 5.4, energy performance errors between calculated and simulated results of the IPM, original BB and our NLP-BB algorithm are 2.76%, 2.18% and 2.47%, respectively.

Their time performance errors are 0.78%, 0.5% and 0.61%, respectively. Performance errors exist in all metrics but remain trivial, leading to accurate and reliable calculated results of our method for early-stage performance prediction.

A new question is raised which is why in all cases the simulated results of the time metric are less than the calculated results while the simulated results of the energy metric are more than calculated results. It is because leakage energy is recorded in the simulated results but not included in the calculated results. Since the baseline metrics in our models involve more behaviours (like data storage to IP cores) than in the simulations, larger calculated results should be given in both energy and time performance. But the Orion model in NIRGAM records the leakage energy of the network during simulations while our models do not. The leakage energy increases along with the simulation time, which overwhelms the extra calculated values. Thus larger energy and smaller time simulated results over calculated results are achieved, suggesting the potential solution of adding a leakage model in our method to increase accuracy.

Compared to the synthetic application performance, our method solves the media application with reduced performance improvements. This is because a small-size network is chosen to provide a relatively small searching pool (potential solution set) for the mapping generation of all algorithms. It is believed that as the network size and application complexity increases, the IPM and original BB algorithms have a higher chance to cost longer time for optimal mappings or performance drop with limited time than our NLP-BB algorithm, indicating the superiority of our algorithm in balancing the result optimality and execution efficiency.

5.4 Summary

Finding proper task mappings of complex applications with satisfactory performance and high-efficiency execution is hard to balance. In this chapter we have proposed a novel mapping method balancing the result optimality and execution efficiency of produced mappings. This mapping technique contributes a more intelligent and generalised mapping algorithm to the task allocation step of current NoC design methodology as shown in Figure 1.2, which improves the feasibility of generating optimised NoC systems for subsequent implementation. Non-Linear Programming (NLP) based problem formulation and a modified branch and bound (BB) algorithm are developed in the proposed task allocation method, providing topological extensibility, global optimality and high-efficiency execution for mapping generation. More accurate energy and timing models are designed for precise calculations. Experiments reveal that the mappings produced by our method have competitive performance results in terms of energy and time and superior execution efficiency, comparing to an exact NLP algorithm, Interior Point Method (IPM), and the original BB algorithm. Specifically, our mapping achieved around $(9.28 \sim 30.87)\%$ performance savings and about $(48.46 \sim 102.73)$ times faster execution speed than IPM, a slight performance drop of only about $(1 \sim 4)\%$ than original BB but with a $(3 \sim 20)$-fold speedup in execution. The accuracy of calculated performance results in our method also verified that minor errors $(2\% \sim 4\%)$ are found in comparison with simulated results.

Apart from offering a better balance between mapping performance and execution efficiency, our work also reports potential directions for extension. The energy and timing models used in our method can be more accurate by integrating the leakage energy part, making the calculated performance results more precise. The buffer space overhead of our modified BB algorithm is also expensive due to the large storage requirement for candidate mappings at each level of the searching tree. This inspires the future work of developing new policies to ease. The intrinsic exhaustive searching mechanism prevents the BB algorithm from achieving high-efficiency execution to very large-scale applications, also leaving design space of efficient mechanisms to balance the mapping optimality and execution efficiency.

Chapter 6

Conclusions and Future Work

6.1 Summary of the thesis

Today, complex on-chip digital systems are extensively used in products of our daily lives. New system architectures and functional modules are constantly updated to adapt to the fast scaling of system feature size. Network on Chip is one promising architecture for communication-centric system designs. It deploys many single processing units and interconnects them with routers and switches to form a network for collaboratively tackling extremely complex computations in a distributed way. Yet the traditional gate-level synthesis design flow is less suitable to design such highly complex on-chip systems with tolerable design time cycles and prototyping cost, which severely lowers the design productivity. To alleviate the conflict between the reduced design productivity and increased design complexity, electronic design automation techniques are used to aid system designs at higher levels, such that accurate behavioural models of NoC architectures and functional models of performance estimates are developed at system level for achieving accurate early-stage performance and functional evaluations.

As an emerging technique, system-level design automation of NoC architectures leaves many open space for further exploration. The accuracy of abstract high-level models and performance models are a particularly significant issue that directly determines the usefulness of the designs. High applicability of the automation techniques to various specific NoC topologies has also drawn the interest of designers and developers since many custom NoC architectures are needed to meet specific design requirements. Besides, an intelligent task-mapping scheme plays a critically important role in dealing with complex computational applications with a full use of available network resources to generate optimised performance trade-offs. This thesis has addressed those issues to improve the functionality of system-level design automation techniques. Through the work introduced from Chapter 3 to Chapter 5, this thesis has fulfilled the main research objectives listed in Chapter 1

6.1.1 Accurate Model Abstraction of System-Level NoC Architectures

As described in Chapter 3, we have proposed a case study to evaluate the model accuracy of system-level NoC behavioural abstraction and performance estimates. A backbone NoC simulator, NIRGAM, is used for the system-level modelling. In the case study, a system-level asynchronous FIFO architecture has been developed in NIRGAM. This asynchronous FIFO not only extends the functionality of the simulator, but also provides a basic functional test for modelling asynchronous behaviours at system level. Moreover, a small one-to-one data transmission system using the asynchronous FIFO as a buffer is constructed to implement a classic computational application both in NIRGAM and by a RTL synthesis design flow. Comparative experiments have been given at both levels to estimate the performance in terms of timing and power for functional validation of system-level models. The measurement models in NIRGAM is also calibrated from the RTL cell library to improve the performance accuracy. Potential directions of improving the high-level abstract models are investigated based on the result analysis.

Results have shown that the high-level abstract models abstracted by advanced EDA techniques have high precision (less than 4% error to the RTL design) and accurate performance estimates with no jitter insertion, suggesting a superiority of such accurate models to the RTL design in terms of design time cycles and prototyping cost. This system-level modelling would significantly improve the design efficiency and lower the overall design cost of complex on-chip systems. Results have also shown that the NIRGAM simulator lack clock jitter models, which makes the simulator unable to elaborate system behaviours under jitter insertion. Moreover, the power models in NIRGAM is not as thorough as the RTL cell library, leading to a narrower evaluation range. These functional deficiencies have indicated two potential directions of improving the high-level modelling, which are developing more high-level abstractions of general NoC components and developing more accurate model abstractions for current systems.

6.1.2 Efficient Design Method of Application-Specific Network Topologies

The default NoC topological platforms in the NIRGAM simulator are only Mesh and Torus, though they are popular topologies widely used in the current NoC research. Yet there are many other regular, non-rectangular or irregular network topologies which are increasingly popular in various user-specific applications but lack supports from modern system-level design automation. The conventional way of constructing such networks requires custom and heterogenous designs of specific functional components, which prolongs the design cycles and thus lowers the design efficiency.

To fill this gap, a novel method that efficiently emulates virtual non-rectangular and irregular NoC topologies based on an regular mesh network in the NIRGAM simulator has been developed and validated in Chapter 4. In the novel method, a time-regulated model is designed and attached to certain nodes of the Mesh network, changing it to a heterogeneous network. By using this model, the clocks at node output channels can be configured by users to tune the time span of data packets cost at nodes, which then regulates the data traversal time between certain node pairs to virtually form specific routing patterns. Moreover, we have modified the original Orion power model integrated in NIRGAM to provide energy performance estimation for functional validation.

Specifically, two typical non-rectangular example topologies: a honeycomb hexagon and a sparse-octagon, and two irregular routing geometries are developed by our method. Real networks of those non-rectangular and irregular topologies are also customised by the conventional way for functional comparisons. Experiments with data routings along specific geometries are implemented to validate the functionality of the proposed method. Two synthetic traffic scenarios, Uniform Random and Hotspot, are implemented on both virtual and real networks to evaluate the network characteristics and performance. A typical multimedia application, an MPEG-4 decoder, is also given to exploit the potential of our method for performing real-world applications. Limits and problems of our emulation model are summarised based on the result analysis. Their potential solutions are discussed at the end of Chapter 4.

6.1.3 Efficient Optimisation of Task-Mapping Performance Trade-offs in Terms of Timing and Energy

In Chapter 5 we have designed an intelligent task mapping method to tackle complex application tasks with full use of available NoC resources, and optimising the performance trade-offs in terms of energy and timing issues. Besides, based on our proposed mapping method, a performance prediction mechanism of NoC architectures in terms of timing and energy has also been established to offer precise performance evaluations of system-level abstract models. The prediction is expected as the reliable references for the further design stages to help improve the design time cycles and productivity.

For the precise performance prediction, the design specification of our task-mapping problems is to minimise the overall communication energy and timing performance of complex computational applications running on various NoC topologies. The precision and efficiency of the mapping results are significant for performance evaluations at the early design stages. Since the optimisation of task mappings has proved to be an NP-hard problem, it is hard to develop a generalised mapping algorithm that is suitable for all different conditions and can give solutions within an acceptable execution time. Hence, we are motivated develop a new mapping technique to meet our own design specifications. Specifically, we have used a novel Non-linear programming technique to formulate the mapping problem, and a modified branch and bound (BB) algorithm to search the optimal mapping solution with increased execution speed. The

NLP mathematical technique formulates a mapping objective function based on the coordinates of node interrelations instead of the connecting directions between tasks that is limited to be used in certain architectures, which enhances the adaptability of the proposed mapping method to more NoC architectures as long as the nodes of those NoC architectures can be elaborated by a coordinate system. This presents the better functionality of our proposed method than previous mapping algorithms that were developed specifically for certain topologies.

We have also modified a BB algorithm with a refined searching mechanism. It searches the optimal mapping in the more intelligent boundaries of possible mappings, such that the execution time of the algorithm to solve complex computations is explicitly reduced. This modification can remedy the deficiency of the BB algorithm in searching the mapping optimum of large-scale applications with intolerable CPU execution time. The reason for using the BB algorithm in our proposed method is due to its searching mechanism that search the globally optimal mapping from a complete candidate mapping set, which meets our demand for the accurate mapping optimum. Both synthetic and real-world applications are evaluated to validate the functionality of our proposed mapping method. Besides, baseline performance metrics in terms of energy and timing are configured to automatically calculate the performance evaluations in the proposed method. Since both system-level model abstraction and performance prediction are implemented precisely at an early design stage, the proposed method has indicated advanced design automation techniques to offer reliable performance references of NoC applications for the subsequent NoC design stages, contributing to an efficient design step. The precision of performance prediction provided by our proposed method has also been validated in both synthetic and real-media applications.

6.2 Future Work

For the system-level abstract modelling, large design space is still remaining unexplored though the energy and timing models are accurately developed in this thesis. Fault tolerance of NoC applications is a typical concern in NoC architectures. As the system complexity is increasing, more faults may occur for many reasons. Developing high-level fault models for NoC architectures becomes highly meaningful to evaluate the network behaviours. Moreover, detecting potential network faults at the early design stages will dramatically save the design time and cost to revise them. It also supports the mechanism development of fault tolerance and fault recovery. Hence, developing precise system-level fault models in the NIRGAM simulator is part of our future work. Such models can be integrated with existing models to collaboratively abstract precise network failure at the early design stages, facilitating the further research for NoC architectures.

The efficient topology emulation proposed in this thesis needs more design exploration since our proposed method only concerns two performance metrics in terms of energy and timing, though they are of particular importance to on-chip systems. If fault models or fault-tolerant functions can be built up at system-level NoC architectures, our method needs to be modified and extended to emulate such topologies correctly. Specific models for fault detection in our virtual modelling are also required for behavioural research and performance estimates. Besides, more special but popular topologies can be virtually emulated to enhance the functionality of our method. Popular topologies like spidergon, fat tree and 3D networks are used more frequently in modern complex network systems. But the supports of their high-level modelling and performance evaluations are inadequate, which leaves design space to explore in the near future.

Our mapping technique has shown its superiority in the adaptability to diverse NoC architectures and the guaranteed optimality from a global mapping set within tolerable execution time. However, the execution efficiency of the mapping algorithm is still insufficient for very large-scale applications due to the intrinsic exhaustive searching mechanism of the Branch and Bound algorithm. Hence, more intelligent searching mechanisms are needed to detect the potential solutions more efficiently, or in a reasonably reduced range while ensuring the global optimality. Besides, a new mechanism for deadlock-free task placing directions may also be needed since our current XY routing mechanism may be less efficient and less intelligent in complex NoC architectures that have more complex node relations (like 3D networks). An adaptive routing mechanism may be more desirable than deterministic routing mechanism to ensure sufficient use of available resources, if their algorithmic complexity is tolerable for efficient searching and task placement in those complex NoC architectures.

Appendix A

Technology Parameters Used by Orion

```
//Parameters from SIM_technology.h

#define PARM_AF (5.000000e-01)
#define PARM_MAXN (8)
#define PARM_MAXSUBARRAYS (8)
#define PARM_MAXSPD (8)
#define PARM_VTHOUTDRNOR (4.310000e-01)
#define PARM_VTHCOMPINV (4.370000e-01)
#define PARM_BITOUT (64)
#define PARM_ruu_issue_width (4)
#define PARM_amp_Idsat (5.000000e-04)
#define PARM_VSINV (4.560000e-01)
#define PARM_GEN_POWER_FACTOR (1.310000e+00)
#define PARM_VTHNAND60x90 (5.610000e-01)
#define PARM_FUDGEFACTOR (1.000000e+00)
#define PARM_VTHOUTDRIVE (4.250000e-01)
#define PARM_VTHMUXDRV1 (4.370000e-01)
#define PARM_VTHMUXDRV2 (4.860000e-01)
#define PARM_NORMALIZE_SCALE (6.488730e-10)
#define PARM_VTHMUXDRV3 (4.370000e-01)
#define PARM_ADDRESS_BITS (64)
#define PARM_RUU_size (16)
#define PARM_VTHNOR12x4x1 (5.030000e-01)
#define PARM_VTHNOR12x4x2 (4.520000e-01)
#define PARM_VTHOUTDRINV (4.370000e-01)
#define PARM_VTHNOR12x4x3 (4.170000e-01)
#define PARM_VTHEVALINV (2.670000e-01)
#define PARM_VTHNOR12x4x4 (3.900000e-01)
#define PARM_res_ialu (4)
#define PARM_VTHOUTDRNAND (4.410000e-01)
#define PARM_VTHINV100x60 (4.380000e-01)

#if (PARM(TECH_POINT) <= 90 )
#if (PARM(TRANSISTOR_TYPE) == LVT)
#define PARM_Cgatepass (1.5225000e-14)
#define PARM_Cpdiffarea (6.05520000e-15)
#define PARM_Cpdiffside (2.38380000e-15)
#define PARM_Cndiffside (2.8500000e-16)
```

```
#define PARM_Cndiffarea (5.7420000e-15)
#define PARM_Cnoverlap (1.320000e-16)
#define PARM_Cpoverlap (1.210000e-16)
#define PARM_Cgate (7.8648000e-14)
#define PARM_Cpdiffovlp (1.420000e-16)
#define PARM_Cndiffovlp (1.420000e-16)
#define PARM_Cnoxideovlp (2.580000e-16)
#define PARM_Cpoxideovlp (3.460000e-16)

#elif (PARM(TRANSISTOR_TYPE) == NVT)
#define PARM_Cgatepass (8.32500e-15)
#define PARM_Cpdiffarea (3.330600e-15)
#define PARM_Cpdiffside (1.29940000e-15)
#define PARM_Cndiffside (2.5500000e-16)
#define PARM_Cndiffarea (2.9535000e-15)
#define PARM_Cnoverlap (1.270000e-16)
#define PARM_Cpoverlap (1.210000e-16)
#define PARM_Cgate (3.9664000e-14)
#define PARM_Cpdiffovlp (1.31000e-16)
#define PARM_Cndiffovlp (1.310000e-16)
#define PARM_Cnoxideovlp (2.410000e-16)
#define PARM_Cpoxideovlp (3.170000e-16)

#elif  (PARM(TRANSISTOR_TYPE) == HVT)
#define PARM_Cgatepass (1.45000e-15)
#define PARM_Cpdiffarea (6.06000e-16)
#define PARM_Cpdiffside (2.150000e-16)
#define PARM_Cndiffside (2.25000e-16)
#define PARM_Cndiffarea (1.650000e-16)
#define PARM_Cnoverlap (1.220000e-16)
#define PARM_Cpoverlap (1.210000e-16)
#define PARM_Cgate (6.8000e-16)
#define PARM_Cpdiffovlp (1.20000e-16)
#define PARM_Cndiffovlp (1.20000e-16)
#define PARM_Cnoxideovlp (2.230000e-16)
#define PARM_Cpoxideovlp (2.880000e-16)

#endif /*PARM(TRANSISTOR_TYPE) */
#endif /*PARM(TECH_POINT)*/


//Parameters from  "SIM_technology.h

#define PARM_AF (5.000000e-01)
#define PARM_MAXN (8)
#define PARM_MAXSUBARRAYS (8)
#define PARM_MAXSPD (8)
#define PARM_VTHOUTDRNOR (4.310000e-01)
#define PARM_VTHCOMPINV (4.370000e-01)
#define PARM_BITOUT (64)
#define PARM_ruu_issue_width (4)
#define PARM_amp_Idsat (5.000000e-04)
#define PARM_VSINV (4.560000e-01)
#define PARM_GEN_POWER_FACTOR (1.310000e+00)
#define PARM_VTHNAND60x90 (5.610000e-01)
#define PARM_FUDGEFACTOR (1.000000e+00)
#define PARM_VTHOUTDRIVE (4.250000e-01)
#define PARM_VTHMUXDRV1 (4.370000e-01)
#define PARM_VTHMUXDRV2 (4.860000e-01)
```

```
#define PARM_NORMALIZE_SCALE (6.488730e-10)
#define PARM_VTHMUXDRV3 (4.370000e-01)
#define PARM_ADDRESS_BITS (64)
#define PARM_RUU_size (16)
#define PARM_VTHNOR12x4x1 (5.030000e-01)
#define PARM_VTHNOR12x4x2 (4.520000e-01)
#define PARM_VTHOUTDRINV (4.370000e-01)
#define PARM_VTHNOR12x4x3 (4.170000e-01)
#define PARM_VTHEVALINV (2.670000e-01)
#define PARM_VTHNOR12x4x4 (3.900000e-01)
#define PARM_res_ialu (4)
#define PARM_VTHOUTDRNAND (4.410000e-01)
#define PARM_VTHINV100x60 (4.380000e-01)

#if (PARM(TECH_POINT) <= 90 )
#if (PARM(TRANSISTOR_TYPE) == LVT)
#define PARM_Cgatepass (1.5225000e-14)
#define PARM_Cpdiffarea (6.05520000e-15)
#define PARM_Cpdiffside (2.38380000e-15)
#define PARM_Cndiffside (2.8500000e-16)
#define PARM_Cndiffarea (5.7420000e-15)
#define PARM_Cnoverlap (1.320000e-16)
#define PARM_Cpoverlap (1.210000e-16)
#define PARM_Cgate (7.8648000e-14)
#define PARM_Cpdiffovlp (1.420000e-16)
#define PARM_Cndiffovlp (1.420000e-16)
#define PARM_Cnoxideovlp (2.580000e-16)
#define PARM_Cpoxideovlp (3.460000e-16)

#elif (PARM(TRANSISTOR_TYPE) == NVT)
#define PARM_Cgatepass (8.32500e-15)
#define PARM_Cpdiffarea (3.330600e-15)
#define PARM_Cpdiffside (1.29940000e-15)
#define PARM_Cndiffside (2.5500000e-16)
#define PARM_Cndiffarea (2.9535000e-15)
#define PARM_Cnoverlap (1.270000e-16)
#define PARM_Cpoverlap (1.210000e-16)
#define PARM_Cgate (3.9664000e-14)
#define PARM_Cpdiffovlp (1.31000e-16)
#define PARM_Cndiffovlp (1.310000e-16)
#define PARM_Cnoxideovlp (2.410000e-16)
#define PARM_Cpoxideovlp (3.170000e-16)

#elif  (PARM(TRANSISTOR_TYPE) == HVT)
#define PARM_Cgatepass (1.45000e-15)
#define PARM_Cpdiffarea (6.06000e-16)
#define PARM_Cpdiffside (2.150000e-16)
#define PARM_Cndiffside (2.25000e-16)
#define PARM_Cndiffarea (1.650000e-16)
#define PARM_Cnoverlap (1.220000e-16)
#define PARM_Cpoverlap (1.210000e-16)
#define PARM_Cgate (6.8000e-16)
#define PARM_Cpdiffovlp (1.20000e-16)
#define PARM_Cndiffovlp (1.20000e-16)
#define PARM_Cnoxideovlp (2.230000e-16)
#define PARM_Cpoxideovlp (2.880000e-16)

#endif /*PARM(TRANSISTOR_TYPE) */
#endif /*PARM(TECH_POINT)*/
```

```
//Parameters from SIM_technology_v2.h

#if ( PARM(TECH_POINT) <= 90 )
#define Vbitpre     (Vdd)
#define Vbitsense   (0.08)
#define SensePowerfactor3 (CLK_FREQ * 1e9)*(Vbitsense)*(Vbitsense)
#define SensePowerfactor2 (CLK_FREQ * 1e9)*(Vbitpre-Vbitsense)*(Vbitpre-Vbitsense)
#define SensePowerfactor  (CLK_FREQ * 1e9)*Vdd*(Vdd/2)
#define SenseEnergyFactor (Vdd*Vdd/2)
#endif /* PARM(TECH_POINT) */

#if(PARM(TECH_POINT) == 90)
#define LSCALE 0.125
#define MSCALE  (LSCALE * .624 / .2250)
/* bit width of RAM cell in um */
#define BitWidth    (2.24)
/* bit height of RAM cell in um */
#define BitHeight   (2.52)
#define Cout        (6.25e-14)
#define BitlineSpacing   1.12
#define WordlineSpacing  1.12
#define RegCellHeight    2.8
#define RegCellWidth     1.9
#define Cwordmetal   (1.936e-15)
#define Cbitmetal    (3.872e-15)
#define Cmetal        (Cbitmetal/16)
#define CM2metal      (Cbitmetal/16)
#define CM3metal      (Cbitmetal/16)
/* minimal spacing metal cap per unit length */
#define CCmetal       (0.18608e-15)
#define CCM2metal     (0.18608e-15)
#define CCM3metal     (0.18608e-15)
/* 2x minimal spacing metal cap per unit length */
#define CC2metal      (0.12529e-15)
#define CC2M2metal    (0.12529e-15)
#define CC2M3metal    (0.12529e-15)
/* 3x minimal spacing metal cap per unit length */
#define CC3metal      (0.11059e-15)
#define CC3M2metal    (0.11059e-15)
#define CC3M3metal    (0.11059e-15)
/* corresponds to clock network*/
#define Clockwire (404.8e-12)
#define Reswire (36.66e3)
#define invCap (3.816e-14)
#define Resout (213.6)
/* um */
#define Leff          (0.1)
/* length unit in um */
#define Lamda         (Leff * 0.5)
/* fF/um */
#define Cpolywire   (2.6317875e-15)
/* ohms*um of channel width */
#define Rnchannelstatic (3225)
/* ohms*um of channel width */
#define Rpchannelstatic (7650)

#if( PARM(TRANSISTOR_TYPE) == LVT)
```

```
#define Rnchannelon (1716)
#define Rpchannelon (4202)
#elif (PARM(TRANSISTOR_TYPE) == NVT)
#define Rnchannelon (4120)
#define Rpchannelon (10464)
#elif (PARM(TRANSISTOR_TYPE) == HVT)
#define Rnchannelon (4956)
#define Rpchannelon (12092)
#endif

#define Rbitmetal   (1.38048)
#define Rwordmetal   (0.945536)

#if(PARM(TRANSISTOR_TYPE) == LVT)
#define Vt      0.237
#elif(PARM(TRANSISTOR_TYPE) == NVT)
#define Vt      0.307
#elif (PARM(TRANSISTOR_TYPE) == HVT)
#define Vt      0.482
#endif

/* transistor widths in um */
#if(PARM(TRANSISTOR_TYPE) == LVT)
#define Wdecdrivep  (12.50)
#define Wdecdriven  (6.25)
#define Wdec3to8n   (11.25)
#define Wdec3to8p   (7.5)
#define WdecNORn    (0.30)
#define WdecNORp    (1.12)
#define Wdecinvn    (0.84)
#define Wdecinvp    (1.12)
#define Wdff        (12.32)
#define Wworddrivemax   (12.50)
#define Wmemcella   (0.35)
#define Wmemcellr   (0.50)
#define Wmemcellw   (0.26)
#define Wmemcellbscale  (2)
#define Wbitpreequ  (1.25)
#define Wbitmuxn    (1.25)
#define WsenseQ1to4 (0.55)
#define Wcompinvp1  (1.25)
#define Wcompinvn1  (0.84)
#define Wcompinvp2  (2.24)
#define Wcompinvn2  (1.68)
#define Wcompinvp3  (5.15)
#define Wcompinvn3  (3.25)
#define Wevalinvp   (2.24)
#define Wevalinvn   (9.45)
#define Wcompn      (1.25)
#define Wcompp      (3.75)
#define Wcomppreequ     (5.15)
#define Wmuxdrv12n  (3.75)
#define Wmuxdrv12p  (6.25)
#define WmuxdrvNANDn    (2.52)
#define WmuxdrvNANDp    (10.33)
#define WmuxdrvNORn (7.56)
#define WmuxdrvNORp (10.36)
#define Wmuxdrv3n   (24.85)
#define Wmuxdrv3p   (60.25)
```

```
#define Woutdrvseln (1.55)
#define Woutdrvselp (2.33)
#define Woutdrvnandn    (3.36)
#define Woutdrvnandp    (1.4)
#define Woutdrvnorn (0.75)
#define Woutdrvnorp (5.32)
#define Woutdrivern (6.16)
#define Woutdriverp (9.77)
#define Wbusdrvn    (6.16)
#define Wbusdrvp    (10.57)
#define Wcompcellpd2    (0.33)
#define Wcompdrivern    (50.95)
#define Wcompdriverp    (102.67)
#define Wcomparen2      (5.13)
#define Wcomparen1      (2.5)
#define Wmatchpchg      (1.25)
#define Wmatchinvn      (1.4)
#define Wmatchinvp      (3.08)
#define Wmatchnandn     (2.24)
#define Wmatchnandp     (1.68)
#define Wmatchnorn      (2.52)
#define Wmatchnorp      (1.12)
#define WSelORn         (1.4)
#define WSelORprequ     (5.04)
#define WSelPn          (1.86)
#define WSelPp          (1.86)
#define WSelEnn         (0.63)
#define WSelEnp         (1.25)
#define Wsenseextdrv1p  (5.15)
#define Wsenseextdrv1n  (3.05)
#define Wsenseextdrv2p  (25.20)
#define Wsenseextdrv2n  (15.65)

#elif(PARM(TRANSISTOR_TYPE) == NVT)
#define Wdecdrivep  (11.57)
#define Wdecdriven  (5.74)
#define Wdec3to8n   (10.31)
#define Wdec3to8p   (6.87)
#define WdecNORn    (0.28)
#define WdecNORp    (1.4)
#define Wdecinvn    (0.58)
#define Wdecinvp    (1.12)
#define Wdff        (6.72)
#define Wworddrivemax   (11.57)
#define Wmemcella   (0.33)
#define Wmemcellr   (0.46)
#define Wmemcellw   (0.24)
#define Wmemcellbscale  (2)
#define Wbitpreequ  (1.15)
#define Wbitmuxn    (1.15)
#define WsenseQ1to4 (0.49)
#define Wcompinvp1  (1.12)
#define Wcompinvn1  (0.84)
#define Wcompinvp2  (2.24)
#define Wcompinvn2  (1.38)
#define Wcompinvp3  (4.48)
#define Wcompinvn3  (2.88)
#define Wevalinvp   (2.29)
#define Wevalinvn   (8.89)
```

```
#define Wcompn       (1.15)
#define Wcompp       (3.44)
#define Wcomppreequ     (4.66)
#define Wmuxdrv12n  (3.44)
#define Wmuxdrv12p  (5.74)
#define WmuxdrvNANDn    (2.52)
#define WmuxdrvNANDp    (9.24)
#define WmuxdrvNORn (6.72)
#define WmuxdrvNORp (9.49)
#define Wmuxdrv3n   (22.83)
#define Wmuxdrv3p   (55.09)
#define Woutdrvseln (1.40)
#define Woutdrvselp (2.21)
#define Woutdrvnandn    (2.80)
#define Woutdrvnandp    (1.12)
#define Woutdrvnorn (0.69)
#define Woutdrvnorp (4.76)
#define Woutdrivern (5.58)
#define Woutdriverp (9.05)
#define Wbusdrvn    (5.58)
#define Wbusdrvp    (9.45)
#define Wcompcellpd2    (0.29)
#define Wcompdrivern    (46.28)
#define Wcompdriverp    (92.94)
#define Wcomparen2      (4.65)
#define Wcomparen1      (2.29)
#define Wmatchpchg      (1.15)
#define Wmatchinvn      (1.12)
#define Wmatchinvp      (2.52)
#define Wmatchnandn     (2.24)
#define Wmatchnandp     (1.40)
#define Wmatchnorn      (2.37)
#define Wmatchnorp      (1.12)
#define WSelORn         (1.15)
#define WSelORprequ     (4.66)
#define WSelPn          (1.45)
#define WSelPp          (1.71)
#define WSelEnn         (0.58)
#define WSelEnp         (1.15)
#define Wsenseextdrv1p  (4.66)
#define Wsenseextdrv1n  (2.78)
#define Wsenseextdrv2p  (23.02)
#define Wsenseextdrv2n  (14.07)

#elif(PARM(TRANSISTOR_TYPE) == HVT)
#define Wdecdrivep  (10.64)
#define Wdecdriven  (5.23)
#define Wdec3to8n   (9.36)
#define Wdec3to8p   (6.24)
#define WdecNORn    (0.25)
#define WdecNORp    (1.25)
#define Wdecinvn    (0.52)
#define Wdecinvp    (1.04)
#define Wdff        (5.43)
#define Wworddrivemax   (10.64)
#define Wmemcella   (0.25)
#define Wmemcellr   (0.42)
#define Wmemcellw   (0.22)
#define Wmemcellbscale  (2)
```

```
#define Wbitpreequ  (1.04)
#define Wbitmuxn    (1.04)
#define WsenseQ1to4 (0.42)
#define Wcompinvp1  (1.08)
#define Wcompinvn1  (0.62)
#define Wcompinvp2  (2.08)
#define Wcompinvn2  (1.25)
#define Wcompinvp3  (4.16)
#define Wcompinvn3  (2.52)
#define Wevalinvp   (2.08)
#define Wevalinvn   (8.32)
#define Wcompn      (1.04)
#define Wcompp      (3.12)
#define Wcomppreequ     (4.16)
#define Wmuxdrv12n  (3.12)
#define Wmuxdrv12p  (5.23)
#define WmuxdrvNANDn    (2.08)
#define WmuxdrvNANDp    (8.40)
#define WmuxdrvNORn (6.16)
#define WmuxdrvNORp (8.12)
#define Wmuxdrv3n   (20.80)
#define Wmuxdrv3p   (49.92)
#define Woutdrvseln (1.25)
#define Woutdrvselp (2.08)
#define Woutdrvnandn    (2.52)
#define Woutdrvnandp    (1.04)
#define Woutdrvnorn (0.62)
#define Woutdrvnorp (4.16)
#define Woutdrivern (4.99)
#define Woutdriverp (8.32)
#define Wbusdrvn    (4.99)
#define Wbusdrvp    (8.32)
#define Wcompcellpd2    (0.25)
#define Wcompdrivern    (41.60)
#define Wcompdriverp    (83.20)
#define Wcomparen2      (4.16)
#define Wcomparen1      (2.08)
#define Wmatchpchg      (1.04)
#define Wmatchinvn      (1.04)
#define Wmatchinvp      (2.08)
#define Wmatchnandn     (2.08)
#define Wmatchnandp     (1.08)
#define Wmatchnorn      (2.08)
#define Wmatchnorp      (1.04)
#define WSelORn         (1.04)
#define WSelORprequ     (4.16)
#define WSelPn          (1.04)
#define WSelPp          (1.56)
#define WSelEnn         (0.52)
#define WSelEnp         (1.04)
#define Wsenseextdrv1p  (4.16)
#define Wsenseextdrv1n  (2.50)
#define Wsenseextdrv2p  (20.83)
#define Wsenseextdrv2n  (12.48)
#endif

#define CamCellHeight    (4.095)
#define CamCellWidth     (3.51)
#define MatchlineSpacing (0.75)
```

```
#define TaglineSpacing   (0.75)
#define CrsbarCellHeight 2.94
#define CrsbarCellWidth  2.94
#define krise       (0.5e-10)
#define tsensedata  (0.725e-10)
#define tsensetag   (0.325e-10)
#define tfalldata   (0.875e-10)
#define tfalltag    (0.875e-10)

#endif /* PARM(TECH_POINT) <= 90 */
/*============Above are the parameters for 90nm =======================*/

/*======================PARAMETERS for Link==========================*/
#if(PARM(TECH_POINT) == 90) /* PARAMETERS for Link at 90nm */
#if (WIRE_LAYER_TYPE == LOCAL)
#define WireMinWidth            214e-9
#define WireMinSpacing          214e-9
#define WireMetalThickness      363.8e-9
#define WireBarrierThickness    10e-9
#define WireDielectricThickness 363.8e-9
#define WireDielectricConstant  3.3

#elif (WIRE_LAYER_TYPE == INTERMEDIATE)
#define WireMinWidth            275e-9
#define WireMinSpacing          275e-9
#define WireMetalThickness      467.5e-9
#define WireBarrierThickness    10e-9
#define WireDielectricThickness 412.5e-9
#define WireDielectricConstant  3.3

#elif (WIRE_LAYER_TYPE == GLOBAL)
#define WireMinWidth            410e-9
#define WireMinSpacing          410e-9
#define WireMetalThickness      861e-9
#define WireBarrierThickness    10e-9
#define WireDielectricThickness 779e-9
#define WireDielectricConstant  3.3
#endif /*WIRE_LAYER_TYPE for 90nm*/

#endif /* PARM(TECH_POINT) <= 90 */
/*================================================================*/

/*parameters for insertion buffer for links at 90nm*/
#if(PARM(TECH_POINT) == 90)
#define BufferDriveResistance       5.12594e+03
#define BufferIntrinsicDelay        4.13985e-11

#if(PARM(TRANSISTOR_TYPE) == LVT)
#define BufferInputCapacitance      1.59e-15
#define BufferPMOSOffCurrent        116.2e-09
#define BufferNMOSOffCurrent        52.1e-09
#define ClockCap                    2.7e-14

#elif(PARM(TRANSISTOR_TYPE) == NVT)
#define BufferInputCapacitance      4.7e-15
#define BufferPMOSOffCurrent        67.6e-09
#define BufferNMOSOffCurrent        31.1e-09
#define ClockCap                    1.0e-14
```

```
#elif(PARM(TRANSISTOR_TYPE) == HVT)
#define BufferInputCapacitance      15.0e-15//9.5e-15
#define BufferPMOSOffCurrent        19.2e-09
#define BufferNMOSOffCurrent        10.1e-09
#define ClockCap                    0.3e-15
#endif

#endif /* PARM(TECH_POINT) <= 90 */

/*=====================Parameters for Area===========================*/
#if(PARM(TECH_POINT) == 90)
#define AreaNOR         (4.76)
#define AreaINV         (2.24)
#define AreaAND         (4.48)
#define AreaDFF         (16.23)
#define AreaMUX2        (7.06)
#define AreaMUX3        (11.29)
#define AreaMUX4        (16.93)
#endif /* PARM(TECH_POINT) <= 90 */
```

References

[1] D. Atienza, F. Angiolini, S. Murali, A. Pullini, L. Benini, and G. De Micheli, "Network-on-chip design and synthesis outlook," *INTEGRATION, the VLSI journal*, vol. 41, no. 3, pp. 340–359, 2008.

[2] L. M. Ni, "Issues in Designing Truly Scalable Interconnection Networks," *Parallel Processing, 1996. Proceedings of the 1996 ICPP Workshop on Challenges for*, pp. 74–83, 1996.

[3] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks.* Morgan Kaufmann Publishers, Elsevier, 2004.

[4] M. Gaur and V. Laxmi, "NIRGAM: A Versatile and Scalable Simulation Environment for Network on Chip," *Preprint sumbitted to Journal of Microprocessors and Microsystems*, 2009.

[5] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Design of a Switch for Network on Chip Applications," *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*, vol. 5, pp. 5–217, 2003.

[6] P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance Evaluation and Design Trade-offs for Network-on-Chip Interconnect Architectures," *IEEE Transactions on Computers*, vol. 54, no. 8, pp. 1025–1040, 2005.

[7] S. Pasricha and N. Dutt, *On-chip Communication Architectures: System on Chip Interconnect.* Morgan Kaufmann Publishers, Elsevier, 2010.

[8] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection Networks: An Engineering Approach.* Morgan Kaufmann Publishers, Elsevier, 2003.

[9] C. E. Leiserson, "Fat-trees: Universal Networks for Hardware-efficient Supercomputing," *IEEE Transactions on Computers*, vol. 34, pp. 892–901, 1985.

[10] W. Song and D. Edwards, "Spatial Parallelism in the Routers of Asynchronous On-Chip Networks," *University of Manchester, Ph.D Thesis*, 2011. [Online]. Available: http://apt.cs.manchester.ac.uk/publications/thesis/W_Song11_phd.php

[11] S. W. Keckler, O. A. Olukotun, and H. P. Hofstee, *Multicore Processors and Systems.* Springer Science+Business Media, 2009.

[12] 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, 2000.

[13] M. Cho, M. Lis, K. Shim, M. Kinsy, and S. Devadas, "Path-based, Randomized, Oblivious, Minimal Routing," *Proceedings of the 2nd International Workshop on Network on Chip Architectures*, pp. 23–28, 2009.

[14] Z. Wang and A. Herkersdorf, "Software Performance Simulation Strategies for High-Level Embedded System Design," *Performance Evaluation*, vol. 67, no. 8, pp. 717–739, 2010.

[15] P. K. Sahu and S. Chattopadhyay, "A Survey on Application Mapping Strategies for Network-on-Chip Design," *Journal of Systems Architecture: the EUROMICRO Journal*, vol. 59, no. 1, pp. 60–76, 2013.

[16] C. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons," *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, 2005.

[17] A. Jerraya and W. Wolf, *Multiprocessor Systems-on-Chips.* Morgan Kaufmann Publishers, Elsevier, 2004.

[18] W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proceedings of Design Automation Conference*, pp. 1025–1040, 2001.

[19] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist, "Network on chip: An architecture for billion transistor era," *Proceedings of the IEEE NorChip Conference*, vol. 31, pp. 166–173, 2000.

[20] X. Li and X. Xu, "Techniques for Wireless Sensor Networks," *Beijing Institute of Technology Press, Chinese Version*, 2007.

[21] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, "Noc synthesis flow for customized domain specific multiprocessor systems-on-chip," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, no. 2, pp. 113–129, 2005.

[22] J. Hu and R. Marculescu, "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures," *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pp. 688–693, 2003.

[23] S. Murali, L. Benini, and G. De Micheli, "Mapping and Physical Planning of Networks-on-Chip Architectures with Quality-of-Service Guarantees," *Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific*, vol. 1, pp. 27–32, 2005.

[24] F. Angiolini, P. Meloni, S. Carta, L. Benini, and L. Raffo, "Contrasting a noc and a traditional interconnect fabric with layout awareness," *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, pp. 124–129, 2006.

[25] L. Jain, B. M. Al-Hashimi, M. S. Gaur, V. Laxmi, and A. Narayanan, "NIRGAM: a Simulator for NoC Interconnect Routing and Application Modeling," 2007. [Online]. Available: www.date-conference.com/files/file/10-ubooth/ub-1.4-p04.pdf

[26] É. Cota, A. de Morais Amory, and M. S. Lubaszewski, *Reliability, Availability and Serviceability of Networks-on-chip.* Springer Science & Business Media, 2011.

[27] W. Guo, Z. Guo, and J. Xie, "System on Chip Design and Implementation Methodology," *Beijing Publishing House of Electronic Industry, Chinese Version*, 2008.

[28] W. Wolf, "The Future of Multiprocessor Systems-on-Chips," *Design Automation Conference, 2004. Proceedings. 41st*, pp. 681–685, 2004.

[29] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor System-on-Chip (MPSoC) Technology," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 10, pp. 1701–1713, 2008.

[30] L. Benini and G. De Micheli, "Networks on Chips: a New SoC Paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[31] J. Henkel, W. Wolf, and S. Chakradhar, "On-Chip Networks: A Scalable, Communication-centric Embedded System Design Paradigm," *VLSI Design, 2004. Proceedings. 17th International Conference on*, pp. 845–851, 2004.

[32] J. Kim, D. Park, C. Nicopoulos, N. Vijaykrishnan, and C. R. Das, "Design and Analysis of an NoC Architecture from Performance, Reliability and Energy Perspective," *Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems*, pp. 173–182, 2005.

[33] R. Marculescu, U. Ogras, L. Peh, N. Jerger, and Y. Hoskote, "Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 1, pp. 3–21, 2009.

[34] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain *et al.*, "An 80-Tile Sub-100-w TeraFlops Processor in 65-nm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 29–41, 2008.

[35] F. Clermidy, C. Bernard, R. Lemaire, J. Martin, I. Miro-Panades, Y. Thonnart, P. Vivet, and N. Wehn., "A 477mW NoC-based Digital Baseband for MIMO 4G SDR," *Proc. of IEEE International Solid-State Circuits Conference*, pp. 278–279, 2010.

[36] W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *Distributed Computing*, vol. 1, no. 4, pp. 187–196, 1986.

[37] F. T. Leighton, "New Lower Bound Techniques for VLSI," *Mathematical Systems Theory*, vol. 17, no. 1, pp. 47–70, 1984.

[38] J. Kim, J. Balfour, and W. Dally, "Flattened Butterfly Topology for On-Chip Networks," *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 172–182, 2007.

[39] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip," *VLSI Design, 2004. Proceedings. 17th International Conference on*, pp. 693–696, 2004.

[40] W. Zhang, L. Hou, J. Wang, S. Geng, and W. Wu, "Comparison Research between XY and Odd-Even Routing Algorithm of a 2-Dimension 3X3 Mesh Topology Network-on-Chip," *Intelligent Systems, 2009. GCIS'09. WRI Global Congress on*, vol. 3, pp. 329–333, 2009.

[41] D. Wiklund and D. Liu, "SoCBUS: Switched Network on Chip for Hard Real Time Embedded Systems," *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pp. 8–pp, 2003.

[42] L. G. Roberts, "The Evolution of Packet Switching," *Proceedings of the IEEE*, vol. 66, no. 11, pp. 1307–1313, 1978.

[43] L. Fratta, M. Gerla, and L. Kleinrock, "The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design," *Networks*, vol. 3, no. 2, pp. 97–133, 1973.

[44] J. Duato, A. Robles, F. Silla, and R. Beivide, "A Comparison of Router Architectures for Virtual Cut-through and Wormhole Switching in a NOW Environment," *Parallel Processing, 1999. 13th International and 10th Symposium on Parallel and Distributed Processing, 1999. 1999 IPPS/SPDP. Proceedings*, pp. 240–247, 1999.

[45] P. Kermani and L. Kleinrock, "Virtual Cut-through: A New Computer Communication Switching Technique," *Computer Networks (1976)*, vol. 3, no. 4, pp. 267–286, 1979.

[46] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-queued Switch," *Communications, IEEE Transactions on*, vol. 47, no. 8, pp. 1260–1267, 1999.

[47] W. J. Dally, "Virtual-channel Flow Control," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 3, no. 2, pp. 194–205, 1992.

[48] A. V. de Mello, L. C. Ost, F. G. Moraes, and N. L. V. Calazans, "Evaluation of Routing Algorithms on Mesh based NoCs," *Technical Report, FACULDADE DE INFORMATICA,*, pp. 3–11, 2004.

[49] M. Singhal, "Deadlock Detection in Distributed Systems," *Computer*, vol. 22, no. 11, pp. 37–48, 1989.

[50] J. Duato, "A New Theory of Deadlock-free Adaptive Routing in Wormhole Networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 4, no. 12, pp. 1320–1331, 1993.

[51] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing," *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, pp. 278–287, 1992.

[52] W. J. Dally and C. L. Seitz, "Deadlock-free Message Routing in Multiprocessor Interconnection Networks," *Computers, IEEE Transactions on*, vol. 100, no. 5, pp. 547–553, 1987.

[53] A. Lankes, T. Wild, A. Herkersdorf, S. Sonntag, and H. Reinig, "Comparison of Deadlock Recovery and Avoidance Mechanisms to Approach Message Dependent Deadlocks in On-Chip Networks," *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, pp. 17–24, 2010.

[54] J. Upadhyay, V. Varavithya, and P. Mohapatra, "A Traffic-balanced Adaptive Wormhole Routing Ccheme for Two-Dimensional Meshes," *Computers, IEEE Transactions on*, vol. 46, no. 2, pp. 190–197, 1997.

[55] J. Hu and R. Marculescu, "DyAD: Smart Routing for Networks-on-Chip," *Proceedings of the 41st annual Design Automation Conference*, pp. 260–263, 2004.

[56] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, "Application Specific Routing Algorithms for Networks on Chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, p. 316, 2009.

[57] A. Hansson, K. Goossens, and A. Rădulescu, "A Unified Approach to Mapping and Routing on a Network-on-Chip for Both Best-effort and Guaranteed Service Traffic," *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 75–80, 2005.

[58] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS Architecture and Design Process for Network on Chip," *Journal of Systems Architecture*, vol. 50, no. 2, pp. 105–128, 2004.

[59] J. Liang, S. Swaminathan, and R. Tessier, "aSOC: A Scalable, Single-chip Communications Architecture," *Parallel Architectures and Compilation Techniques, 2000. Proceedings. International Conference on*, pp. 37–46, 2000.

[60] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, vol. 26, no. 2, pp. 62–76, 1993.

[61] W. J. Dally and H. Aoki, "Deadlock-free Adaptive Routing in Multicomputer Networks using Virtual Channels," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 4, no. 4, pp. 466–475, 1993.

[62] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: An Infrastructure for Low Area Overhead Packet-Switching Networks on Chip," *INTEGRATION, the VLSI journal*, vol. 38, no. 1, pp. 69–93, 2004.

[63] R. Mullins, A. West, and S. Moore, "Low-latency Virtual-Channel Routers for On-Chip Networks," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 2, p. 188, 2004.

[64] I. Miro-Panades, F. Clermidy, P. Vivet, and A. Greiner, "Physical Implementation of the DSPIN Network-on-Chip in the FAUST Architecture," *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, pp. 139–148, 2008.

[65] Z. Zhang, A. Greiner, and S. Taktak, "A Reconfigurable Routing Algorithm for a Fault-Tolerant 2D-Mesh Network-on-Chip," *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pp. 441–446, 2008.

[66] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato, "Addressing Manufacturing Challenges with Cost-efficient Fault Tolerant Routing," *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, pp. 25–32, 2010.

[67] D. Bertozzi and L. Benini, "Xpipes: a Network-on-Chip Architecture for Gigascale Systems-on-Chip," *Circuits and Systems Magazine, IEEE*, vol. 4, no. 2, pp. 18–31, 2004.

[68] E. Beigné, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An Asynchronous NoC Architecture Providing Low Latency Service and its Multi-Level Design Framework," *Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on*, pp. 54–63, 2005.

[69] T. Bjerregaard and J. Sparso, "A Router Architecture for Connection-oriented Service Guarantees in the MANGO Clockless Network-on-Chip," *Design, Automation and Test in Europe, 2005. Proceedings*, pp. 1226–1231, 2005.

[70] K. K. Paliwal, J. S. George, N. Rameshan, V. Laxmi, M. S. Gaur, V. Janyani, and R. Narasimhan, "Implementation of QoS Aware Q-Routing Algorithm for Network-on-Chip," *Contemporary Computing*, pp. 370–380, 2009.

[71] N. Rameshan, A. Biyani, M. Gaur, V. Laxmi, and M. Ahmed, "Qos Aware Minimally Adaptive XY Routing for NoC," *17th International Conference on Advanced Computing and Communication (ADCOM), Bangalore, India*, 2009.

[72] L. G. Valiant and G. J. Brebner, "Universal Schemes for Parallel Communication," *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pp. 263–277, 1981.

[73] T. Dumitras and R. Marculescu, "On-Chip Stochastic Communication [SoC Applications]," *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pp. 790–795, 2003.

[74] P. Bogdan and R. Marculescu, "A Theoretical Framework for On-Chip Stochastic Communication Analysis," *Nano-Networks and Workshops, 2006. NanoNet'06. 1st International Conference on*, pp. 1–5, 2006.

[75] Arteris: The Network-on-Chip Interconnect IP Company. [Online]. Available: http://www.arteris.com

[76] Sonics: The Trusted Leader in On-Chip Networks. [Online]. Available: http://sonicsinc.com

[77] The IBM CoreConnect Bus Architecture. [Online]. Available: https://www-01.ibm.com/chips/techlib/techlib.nsf/products/CoreConnect_Bus_Architecture

[78] T. G. Mattson, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl *et al.*, "The 48-core scc processor: the programmer's view," *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, 2010.

[79] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom *et al.*, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pp. 108–109, 2010.

[80] J. Schutz, "A 3.3v 0.6um bicmos superscalar microprocessor," *International Solid-State Circuits Conference (ISSCC), Digest of Technical Papers*, pp. 202–203, 1994.

[81] C. Ramey, "Tile-gx100 manycore processor: Acceleration interfaces and architecture," *Proceedings of the 23th Hot Chips Symposium*, 2011.

[82] EZchip Semiconductor. [Online]. Available: http://www.tilera.com

[83] J. Bammi, E. Harcourt, W. Kruitzer, L. Lavagno, and M. Lazarescu, “Software Performance Estimation Strategies in a System-Level Design Tool,” *Hardware/Software Codesign, 2000. CODES 2000. Proceedings of the Eighth International Workshop on*, pp. 82–86, 2000.

[84] M. Lazarescu, J. Bammi, E. Harcourt, L. Lavagno, and M. Lajolo, “Compilation-based Software Performance Estimation for System Level Design,” *High-Level Design Validation and Test Workshop, 2000. Proceedings. IEEE International*, pp. 167–172, 2000.

[85] P. Gerin, X. Guérin, and F. Pétrot, “Efficient Implementation of Native Software Simulation for MPSoC,” *Design, Automation and Test in Europe, 2008. DATE’08*, pp. 676–681, 2008.

[86] D. Brunelli, L. Benini, C. Moser, and L. Thiele, “An Efficient Solar Energy Harvester for Wireless Sensor Nodes,” *Proceedings of the conference on Design, automation and test in Europe*, pp. 104–109, 2008.

[87] H. S. Wang, X. Zhu, L. S. Peh, and S. Malik, “Orion: a Pwer-performance Simulator for Interconnection Networks,” *MICRO-35*, pp. 294–305, 2002.

[88] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, “Energy Sonservation in Wireless Sensor Networks: A survey,” *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, 2009.

[89] C. Basile, M. Gupta, Z. Kalbarczyk, and R. Iyer, “An Approach for Detecting and Distinguishing Errors versus Attacks in Sensor Networks,” *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pp. 473–484, 2006.

[90] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless Sensor Networks: a Survey,” *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[91] R. G. Michael and S. J. David, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* WH Freeman & Co., San Francisco, 1979.

[92] J. Hu and R. Marculescu, “Energy-aware Mapping for Tile-based NoC Architectures under Performance Constraints,” *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, pp. 233–239, 2003.

[93] P. Ruxandra and K. Shashi, “A Survey of Techniques for Mapping and Scheduling Applications to Network on Chip Systems,” *Research Report of Jonkoping University, School of Engineering*, vol. 4, p. 4, 2004.

[94] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi, “Overhead-conscious Voltage Selection for Dynamic and Leakage Energy Reduction of Time-constrained Systems,” *IEE Proceedings, Computers and Digital Techniques*, vol. 152, no. 1, pp. 28–38, 2005.

[95] T. Lei and S. Kumar, "A Two-step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture," *Digital System Design, 2003. Proceedings. Euromicro Symposium on*, pp. 180–187, 2003.

[96] C.-L. Chou and R. Marculescu, "Incremental Run-Time Application Mapping for Homogeneous NoCs with Multiple Voltage Levels," *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, pp. 161–166, 2007.

[97] C.-L. Chou, U. Y. Ogras, and R. Marculescu, "Energy-and Performance-aware Incremental Mapping for Networks on Chip with Multiple Voltage Levels," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 10, pp. 1866–1879, 2008.

[98] C.-L. Chou and R. Marculescu, "User-aware Dynamic Task Allocation in Networks-on-Chip," *Design, Automation and Test in Europe, 2008. DATE'08*, pp. 1232–1237, 2008.

[99] A. Mehran, A. Khademzadeh, and S. Saeidi, "DSM: A Heuristic Dynamic Spiral Mapping Algorithm for Network on Chip," *IEICE Electronics Express*, vol. 5, no. 13, pp. 464–471, 2008.

[100] A. Mehran, S. Saeidi, A. Khademzadeh, and A. Afzali-Kusha, "Spiral: A heuristic Mapping Algorithm for Network on Chip," *IEICE Electronics Express*, vol. 4, no. 15, pp. 478–484, 2007.

[101] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs," *Rapid System Prototyping, 2007. RSP 2007. 18th IEEE/IFIP International Workshop on*, pp. 34–40, 2007.

[102] E. L. de Souza Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic Task Mapping for MPSoCs," *Design & Test of Computers, IEEE*, vol. 27, no. 5, pp. 26–35, 2010.

[103] A. K. Singh, W. Jigang, A. Prakash, and T. Srikanthan, "Mapping Algorithms for NoC-based Heterogeneous MPSoC Platforms," *Digital System Design, Architectures, Methods and Tools, 2009. DSD'09. 12th Euromicro Conference on*, pp. 133–140, 2009.

[104] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communication-aware Heuristics for Run-Time Task Mapping on NoC-based MPSoC Platforms," *Journal of Systems Architecture*, vol. 56, no. 7, pp. 242–255, 2010.

[105] M. Mandelli, L. Ost, E. Carara, G. Guindani, T. Gouvea, G. Medeiros, and F. G. Moraes, "Energy-aware Dynamic Task Mapping for NoC-based MPSoCs," *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pp. 1676–1679, 2011.

[106] M. Mandelli, A. Amory, L. Ost, and F. G. Moraes, "Multi-task Dynamic Mapping onto NoC-based MPSoCs," *Proceedings of the 24th symposium on Integrated circuits and systems design*, pp. 191–196, 2011.

[107] A. Bender, "MILP based Task Mapping for Heterogeneous Multiprocessor Systems," *Proceedings of Design Automation Conference with EURO-VHDL'96 and Exhibition(EURO-DAC'96)*, pp. 190–197, 1996.

[108] C.-E. Rhee, H.-Y. Jeong, and S. Ha, "Many-to-Many Core-Switch Mapping in 2-D Mesh NoC Architectures," *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on*, pp. 438–443, 2004.

[109] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-Programming-based Techniques for Synthesis of Network-on-Chip Architectures," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 4, pp. 407–420, 2006.

[110] O. Ozturk, M. Kandemir, and S. W. Son, "An ILP Based Approach to Reducing Energy Consumption in NoC Based CMPS," *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pp. 411–414, 2007.

[111] P. Ghosh, A. Sen, and A. Hall, "Energy Efficient Application Mapping to NoC Processing Elements Operating at Multiple Voltage Levels," *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pp. 80–85, 2009.

[112] J. Huang, C. Buckl, A. Raabe, and A. Knoll, "Energy-aware Task Allocation for Network-on-Chip based Heterogeneous Multiprocessor Systems," *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pp. 447–454, 2011.

[113] S. Tosun, O. Ozturk, and M. Ozen, "An ILP Formulation for Application Mapping onto Network-on-Chips," *Application of Information and Communication Technologies, 2009. AICT 2009. International Conference on*, pp. 1–5, 2009.

[114] S. Tosun, "Cluster-based Application Mapping Method for Network-on-Chip," *Advances in Engineering Software*, vol. 42, no. 10, pp. 868–874, 2011.

[115] C.-L. Chou and R. Marculescu, "Contention-aware Application Mapping for Network-on-Chip Communication Architectures," *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, pp. 164–169, 2008.

[116] J. Hu and R. Marculescu, "Energy- and Performance-aware Mapping for Regular NoC Architectures," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 4, pp. 551–562, 2005.

[117] T.-J. Lin, S.-Y. Lin, and A.-Y. Wu, "Traffic-balanced IP Mapping Algorithm for 2D-Mesh On-Chip-Networks," *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pp. 200–203, 2008.

[118] M. Reshadi, A. Khademzadeh, and A. Reza, "Elixir: A New Bandwidth-constrained Mapping for Networks-on-Chip," *IEICE Electronics Express*, vol. 7, no. 2, pp. 73–79, 2010.

[119] S. Murali and G. De Micheli, "Bandwidth-constrained Mapping of Cores onto NoC Architectures," *Proceedings of the conference on Design, automation and test in Europe-Volume 2*, p. 20896, 2004.

[120] W. Zhou, Y. Zhang, and Z. Mao, "An Application Specific NoC Mapping for Optimized Delay," *Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006. International Conference on*, pp. 184–188, 2006.

[121] G. Ascia, V. Catania, and M. Palesi, "Multi-objective Mapping for Mesh-based NoC Architectures," *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 182–187, 2004.

[122] ——, "A Multi-objective Genetic Approach to Mapping Problem on Network-on-Chip," *Journal of Universal Computer Science*, vol. 12, no. 4, pp. 370–394, 2006.

[123] K. Bhardwaj and R. K. Jena, "Energy and Bandwidth Aware Mapping of IPs onto Regular NoC Architectures using Multi-Objective Genetic Algorithms," *System-on-Chip, 2009. SOC 2009. International Symposium on*, pp. 027–031, 2009.

[124] F. Moein-darbari, A. Khademzadeh, and G. Gharooni-fard, "Evaluating the Performance of a Chaos Genetic Algorithm for Solving the Network on Chip Mapping Problem," *Computational Science and Engineering, 2009. CSE'09. International Conference on*, vol. 2, pp. 366–373, 2009.

[125] F. Moein-Darbari, A. Khademzade, and G. Gharooni-Fard, "Cgmap: A New Approach to Network-on-Chip Mapping Problem," *IEICE Electronics Express*, vol. 6, no. 1, pp. 27–34, 2009.

[126] J. Kennedy, "Particle Swarm Optimization," *Encyclopedia of Machine Learning*, pp. 760–766, 2010.

[127] Z. Wenbiao, Y. Zhang, G. Shenzhen, Z. Mao, and H. Harbin, "Link-load Balance Aware Mapping and Routing for NoC," *Architecture*, vol. 4, no. 5, p. 6, 2007.

[128] A. Colorni, M. Dorigo, V. Maniezzo *et al.*, "Distributed Optimization by Ant Colonies," *Proceedings of the first European conference on artificial life*, vol. 142, pp. 134–142, 1991.

[129] J. Wang, Y. Li, S. Chai, and Q. Peng, "Bandwidth-aware Application Mapping for NoC-based MPSoCs," *Journal of Computational Information Systems*, vol. 7, no. 1, pp. 152–159, 2011.

[130] C. Çelik and C. F. Bazlamaçcı, "Evaluation of energy and buffer aware application mapping for networks-on-chip," *Microprocessors and Microsystems*, vol. 38, no. 4, pp. 325–336, 2014.

[131] Q. Le, G. Yang, W. N. Hung, X. Song, and X. Zhang, "Pareto optimal mapping for tile-based network-on-chip under reliability constraints," *International Journal of Computer Mathematics*, vol. 92, no. 1, pp. 41–58, 2015.

[132] N. Koziris, M. Romesis, P. Tsanakas, and G. Papakonstantinou, "An Efficient Algorithm for the Physical Mapping of Clustered Task Graphs onto Multiprocessor Architectures," *Parallel and Distributed Processing, 2000. Proceedings. 8th Euromicro Workshop on*, pp. 406–413, 2000.

[133] S. Saeidi, A. Khademzadeh, and A. Mehran, "SMAP: An Intelligent Mapping Tool for Network on Chip," *Signals, Circuits and Systems, 2007. ISSCS 2007. International Symposium on*, vol. 1, pp. 1–4, 2007.

[134] W.-T. Shen, C.-H. Chao, Y.-K. Lien, and A.-Y. A. Wu, "A New Binomial Mapping and Optimization Algorithm for Reduced-Complexity Mesh-based On-Chip Network," *Proceedings of the First International Symposium on Networks-on-Chip*, pp. 317–322, 2007.

[135] M. Tavanpour, A. Khademzadeh, and M. Janidarmian, "Chain-Mapping for Mesh based Network-on-Chip Architecture," *IEICE Electronics Express*, vol. 6, no. 22, pp. 1535–1541, 2009.

[136] Y. Chen, L. Xie, and J. Li, "An Energy-aware Heuristic Constructive Mapping Algorithm for Network on Chip," *ASIC, 2009. ASICON'09. IEEE 8th International Conference on*, pp. 101–104, 2009.

[137] M. Janidarmian, A. Khademzadeh, and M. Tavanpour, "Onyx: A New Heuristic Bandwidth-constrained Mapping of Cores onto Tile-based Network on Chip," *IEICE Electronics Express*, vol. 6, no. 1, pp. 1–7, 2009.

[138] S. Saeidi, A. Khademzadeh, and F. Vardi, "Crinkle: A Heuristic Mapping Algorithm for Network on Chip," *IEICE Electronics Express*, vol. 6, no. 24, pp. 1737–1744, 2009.

[139] M. Janidarmian, A. Khademzadeh, A. R. Fekr, and V. S. Bokharaei, "Citrine: A Methedology for Application-Specific Network-on-Chips Design," *Proceedings of World Congress on Engineering and Computer Science*, vol. 1, pp. 196–202, 2010.

[140] B. Warneke, M. Last, B. Liebowitz, and K. S. Pister, "Smart Dust: Communicating with a Cubic-Millimeter Computer," *Computer*, vol. 34, no. 1, pp. 44–51, 2001.

[141] J. L. Hill and D. E. Culler, "Mica: A Wireless Platform for Deeply Embedded Networks," *Micro, IEEE*, vol. 22, no. 6, pp. 12–24, 2002.

[142] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research," *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pp. 364–369, 2005.

[143] G. Khan and V. Dumitriu, "A Modeling Tool for Simulating and Design of On-Chip Network Systems," *Microprocessors and Microsystems*, vol. 34, no. 2-4, pp. 84–95, 2010.

[144] H. Gu, "A review of research on network-on-chip simulator," *Communication Systems and Information Technology*, pp. 103–110, 2011.

[145] M. Karl, "A Comparison of the Architecture of Network Simulators NS-2 and Tossim," *Proceedings of Performance Simulation of Algorithms and Protocols Seminar, Universit Stuttgart*, pp. 1–15, 2005.

[146] G. Merrett, "Energy-and Information-Managed Wireless Sensor Networks: Modelling and Simulation," *University of Southampton, Ph.D Thesis*, 2008. [Online]. Available: http://eprints.soton.ac.uk/65002/

[147] Y. Xue, H. Lee, M. Yang, P. Kumarawadu, H. Ghenniwa, and W. Shen, "Performance Evaluation of NS-2 Simulator for Wireless Sensor Networks," *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*, pp. 1372–1375, 2007.

[148] I. Downard, *Simulating sensor networks in ns-2.* DTIC Document, Network and Communication Systems, Information Technology Division, 2004.

[149] S. Park, A. Savvides, and M. Srivastava, "SensorSim: A Simulation Framework for Sensor Networks," *Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pp. 104–111, 2000.

[150] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "A Design Methodology for Application-Specific Networks-on-chip," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 5, no. 2, pp. 263–280, 2006.

[151] L. Bononi and N. Concer, "Simulation and Analysis of Network on Chip Architectures: Ring, Spidergon and 2D Mesh," *Proceedings of the conference on Design, automation and test in Europe: Designers' forum*, pp. 154–159, 2006.

[152] F. Fazzino, M. Palesi, and D. Patti, “Noxim: Network-on-chip simulator,” 2008. [Online]. Available: http://sourceforge.net/projects/noxim

[153] C. Grecu, A. Ivanov, R. Saleh, C. Rusu, L. Anghel, P. P. Pande, and V. Nuca, “A flexible network-on-chip simulator for early design space exploration,” *Microsystems and Nanoelectronics Research Conference, 2008. MNRC 2008. 1st*, pp. 33–36, 2008.

[154] C.-F. Chang and Y. Hsu, “A system exploration platform for network-on-chip,” *Parallel and Distributed Processing with Applications (ISPA), 2010 International Symposium on*, pp. 359–366, 2010.

[155] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, “Garnet: A detailed on-chip network model inside a full-system simulator,” *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pp. 33–42, 2009.

[156] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, “The gem5 simulator,” *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[157] A. Kahng, B. Li, L. Peh, and K. Samadi, “Orion 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration,” *Proceedings of the conference on Design, Automation and Test in Europe*, pp. 423–428, 2009.

[158] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, “× pipescompiler: A tool for instantiating application specific networks on chip,” *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2, pp. 884–889, 2004.

[159] C. Sun, C.-H. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, “Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling,” *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pp. 201–210, 2012.

[160] M. Zwolinski, *Digital System Design with SystemVerilog.* Pearson Education, 2009.

[161] ModelSim. [Online]. Available: http://www.mentor.com/products/fv/modelsim/

[162] Design Compiler. [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages/default.aspx

[163] L. Jain, “NIRGAM manual,” 2007. [Online]. Available: http://nirgam.ecs.soton.ac.uk/Documentation.php

[164] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach.* Elsevier, 2011.

[165] M. Krstić, E. Grass, F. K. Gürkaynak, and P. Vivet, “Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook,” *IEEE Design and Test*, vol. 24, no. 5, pp. 430–441, 2007.

[166] I. R. Committee, “International Technology Roadmap for Semiconductors,” pp. 12–13, 2009. [Online]. Available: www.itrs.net

[167] T.-T. Nguyen and X.-T. Tran, “A novel asynchronous first-in-first-out adapting to multi-synchronous network-on-chips,” *Advanced Technologies for Communications (ATC), 2014 International Conference on*, pp. 365–370, 2014.

[168] E. Kasapaki and J. Sparso, “Argo: A time-elastic time-division-multiplexed noc using asynchronous routers,” *Asynchronous Circuits and Systems (ASYNC), 2014 20th IEEE International Symposium on*, pp. 45–52, 2014.

[169] N. Choudhary, M. Gaur, and V. Laxmi, “Irregular NoC Simulation Framework: IrNIRGAM,” *Emerging Trends in Networks and Computer Communications (ETNCC), 2011 International Conference on*, pp. 1–5, 2011.

[170] W. Chen, D. Jin, and L. Zeng, “Heterogeneous Design Methodology with Configurable Regular Topology Set for Salable Network-on-Chip Designs,” *ASICON’07*, pp. 1293–1296, 2007.

[171] C. Neeb and N. Wehn, “Designing Efficient Irregular Networks for Heterogeneous Systems-on-chip,” *Journal of Systems Architecture*, vol. 54, no. 3-4, pp. 384–396, 2008.

[172] A. W. Yin, T. C. Xu, P. Liljeberg, and H. Tenhunen, “Explorations of Honeycomb Topologies for Network-on-Chip,” *NPC’09*, pp. 73–79, 2009.

[173] A. Yin, N. Chen, P. Liljeberg, and H. Tenhunen, “Comparison of mesh and honeycomb network-on-chip architectures,” *7th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1716–1720, 2012.

[174] R. Gao, F. Liu, H. Gu, and X. Fu, “A double-layer sparse honeycomb topology for NoC,” *3rd International Conference on Computer Science and Network Technology (ICCSNT)*, pp. 689–693, 2013.

[175] F. Karim, A. Nguyen, and S. Dey, “An Interconnect Architecture for Networking Systems on Chips,” *IEEE micro*, vol. 22, no. 5, pp. 36–45, 2002.

[176] M. F. A. Qasem and H. Gu, “Square-octagon interconnection architecture for network-on-chips,” *Signal Processing, Communications and Computing (ICSPCC), 2014 IEEE International Conference on*, pp. 715–719, 2014.

[177] I. Stojmenovic, “Honeycomb Networks: Topological Properties and Communication Algorithms,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 10, pp. 1036–1042, 1997.

[178] P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections," *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 250–256, 2000.

[179] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: a Novel On-chip Communication Network," *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*, p. 15, 2004.

[180] M. Chen, K. Shin, and D. Kandlur, "Addressing, Routing, and Broadcasting in Hexagonal Mesh Multiprocessors," *Computers, IEEE Transactions on*, vol. 39, no. 1, pp. 10–18, 1990.

[181] F. Karim, A. Nguyen, S. Dey, and R. Rao, "On-Chip Communication Architecture for OC-768 Network Processors," *Proceedings of the 38th Annual Design Automation Conference*, pp. 678–683, 2001.

[182] A. Banerjee, P. T. Wolkotte, R. D. Mullins, S. W. Moore, and G. J. Smit, "An energy and performance exploration of network-on-chip architectures," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 3, pp. 319–329, 2009.

[183] G. Ascia, V. Catania, M. Palesi, and D. Patti, "Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip," *Computers, IEEE Transactions on*, vol. 57, no. 6, pp. 809–820, 2008.

[184] XviD codec. [Online]. Available: http://www.xvidmovies.com/codec/

[185] M. Ahmed, M. Gaur, and V. Laxmi, "Adaptive routing over the 2d hexagonal noc," *The International Conference on Embedded Systems (ICES 2010)*, pp. 1–5, 2010.

[186] T. T. Ye, G. D. Micheli, and L. Benini, "Analysis of Power Consumption on Switch Fabrics in Network Routers," *Proceedings of the 39th annual Design Automation Conference*, pp. 524–529, 2002.

[187] G. Ascia, V. Catania, and M. Palesi, "Mapping cores on network-on-chip," *International Journal of Computational Intelligence Research*, vol. 1, no. 1, pp. 109–126, 2005.

[188] F. A. Potra and S. J. Wright, "Interior-point methods," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1, pp. 281–302, 2000.

[189] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, and Z. Wang, "A NoC Traffic Suite based on Real Applications," *VLSI (ISVLSI), 2011 IEEE Computer Society Annual Symposium on*, pp. 66–71, 2011.