Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Network-on-Chip Design for Specific Traffic Pattern

RUICHI HAN

Network-on-Chip Design for Specific Traffic Pattern

RUICHI HAN

Master's Programme, Embedded Systems, 120 credits
Date: June 19, 2026

Supervisors: Yizhi Chen, Zhonghai Lu
Examiner: Ahmed Hemani
School of Electrical Engineering and Computer Science
Swedish title: Nätverk-på-chip-design för specifika trafikmönster

© 2026 Ruichi Han

Abstract

As many-core processors, graphics processing units, and domain-specific accelerators integrate an increasing number of processing elements on a single chip, the on-chip interconnect becomes a critical factor for system performance, area, and scalability. This thesis investigates a complete Network-on-Chip design for sustained permutation-style communication in a 25-node system. The objective is to construct and characterize the design from architectural analysis to register-transfer level implementation, simulation-based verification, logic synthesis, and preliminary physical design.

The proposed architecture uses a 5×5 bidirectional torus topology with deterministic dimension-ordered shortest-path routing. The communication problem is described using a multi-commodity-flow formulation. The injection-control requirement for guaranteed throughput is derived, and the corresponding Network-on-Chip frequency requirements are examined. A local injection-control strategy is selected because it provides similar steady-state behaviour to a centralized controller while reducing hardware and communication costs. The architecture is implemented in SystemVerilog using a single-cycle bufferless router with two parallel channels per direction and a network interface with buffers. A self-checking testbench is developed for functional verification and performance measurement.

Simulation confirms the correctness of end-to-end packet delivery. Static-frequency link-utilization measurements match the analytical model, while dynamic frequency scaling increases average utilization from 20.4% to 26.5% for a fine-grained strategy and to 25.4% for a five-level quantized strategy. Synthesis using a 28-nm standard-cell library shows that area scales almost linearly with payload width, mainly due to the network-interface data queues, router crossbar, and output registers. A preliminary physical design closes timing at approximately 800 MHz post-layout. These results show that a simple bufferless torus Network-on-Chip can support the target permutation-style traffic under the assumptions of this thesis, while providing a baseline for future architectural and physical-design extensions.

Keywords

Network-on-Chip, Guaranteed throughput, Bufferless router, Permutation traffic, Torus topology

Sammanfattning

I takt med att flerkärniga processorer, grafikprocessorer och domänspecifika acceleratorer integrerar ett ökande antal beräkningselement på samma chip blir den interna chipkommunikationen en kritisk faktor för systemets prestanda, area och skalbarhet. Detta examensarbete undersöker en komplett Network-on-Chip-design för varaktig permutationsbaserad kommunikation i ett system med 25 noder. Målet är att konstruera och karakterisera designen från arkitekturanalys till RTL-implementation, simuleringsbaserad verifiering, logiksyntes och preliminär fysisk design.

Den föreslagna arkitekturen använder en 5×5 dubbelriktad torustopologi med deterministisk dimensionsordnad routning längs kortaste väg. Kommunikationsproblemet beskrivs med hjälp av en multikommoditetsflödesformulering. Kravet på injektionskontroll för garanterad genomströmning härleds, och motsvarande frekvenskrav för Network-on-Chip undersöks. En lokal strategi för injektionskontroll väljs eftersom den ger ett liknande stationärt beteende som en centraliserad styrenhet, samtidigt som hårdvaru- och kommunikationskostnaden minskar. Arkitekturen implementeras i SystemVerilog med en encykels bufferlös router med två parallella kanaler per riktning och ett nätverksgränssnitt med buffertar. En självkontrollerande testbänk utvecklas för funktionell verifiering och prestandamätning.

Simuleringen bekräftar korrektheten av paketleveransen från ände till ände. Mätningar av länkutnyttjande vid statisk frekvens överensstämmer med den analytiska modellen, medan dynamisk frekvensskalning ökar det genomsnittliga utnyttjandet från 20.4% till 26.5% för en fingranulär strategi och till 25.4% för en kvantiserad strategi med fem nivåer. Syntes med ett 28-nm standardcellbibliotek visar att arean skalar nästan linjärt med nyttolastbredden, huvudsakligen på grund av nätverksgränssnittets dataköer, routerns korskoppling och utgångsregister. En preliminär fysisk design uppnår timingstängning vid ungefär 800 MHz efter layout. Resultaten visar att ett enkelt bufferlöst torusbaserat Network-on-Chip kan stödja den avsedda permutationsbaserade trafiken under examensarbetets antaganden, samtidigt som det ger en grund för framtida arkitektur- och fysisk-designutveckling.

Nyckelord

Network-on-Chip, garanterad genomströmning, bufferlös router, permutationstrafik, torustopologi

Acknowledgments

The past two years have passed almost in the blink of an eye. It feels as if I was only recently standing in the metro station in Stockholm, confused by the signs and unsure of where to go, and yet now my master's studies are coming to an end. In these two years, many things seem to have changed, yet also seem to have remained the same. In any case, this thesis marks an important conclusion to this stage of my journey.

First, I would like to express my sincere gratitude to Prof. Zhonghai Lu. This thesis was completed under his guidance, and I am deeply grateful for his academic support, trust, and encouragement throughout this work. I would also like to thank Prof. Ahmed Hemani. The courses related to digital design that he taught provided me with an important foundation. The knowledge and training from these courses helped me build the background necessary to complete this thesis.

My special thanks go to Yizhi Chen, my supervisor, who has provided me with academic guidance since last autumn. His support, feedback, and advice have helped me greatly throughout the development of this thesis. I sincerely wish him every success in his future career.

I am also grateful to my family for supporting my decision to study at KTH and for giving me the opportunity to pursue this path. Their understanding and encouragement have been an important source of strength during my time abroad. I would like to thank Tong Lei, Ziyin Yang, and Yurun Chao. They accompanied me throughout my two years of master's study and helped me greatly, both in coursework and in daily life, making these two years much richer and more memorable. I sincerely wish them a bright and successful future.

Finally, I would like to thank myself for completing my master's studies successfully. During my undergraduate years, I once wrote down the words "be happy". Looking back, I am glad that I have indeed lived these two years with that wish in mind. I will continue to carry these words with me and move forward into the next stage of my life.

AI Usage Statement: Generative AI was used only for language polishing, and no scientific content was generated by AI.

Stockholm, June 2026
Ruichi Han

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Problem Definition . . . 2
1.3 Purpose and Goals . . . 3
1.4 Research Methodology . . . 4
1.5 Ethics and Sustainability . . . 5
1.6 Delimitations . . . 7
1.7 Structure of the Thesis . . . 7

2 Background **9**
2.1 Motivation and Scope of Network-on-Chip . . . 9
2.2 Network-on-Chip Hardware Architecture . . . 10
2.2.1 Fundamental components of Network-on-Chip . . . 10
2.2.2 Topology . . . 10
2.3 Routing and Switching . . . 12
2.3.1 Routing Algorithms . . . 12
2.3.2 Switching Modes and Resource Sharing . . . 13
2.4 Related Work . . . 14
2.4.1 Packet Switched NoCs . . . 14
2.4.2 Circuit Switched TDM NoCs . . . 15
2.4.3 Circuit Switched SDM NoCs . . . 17
2.4.4 Bufferless and Minimally Buffered NoCs . . . 17
2.4.5 Recent Large-Scale Interconnect Studies . . . 20

3 Network Design and Analysis **21**
3.1 Multi-Commodity Flow Formulation . . . 21
3.2 Topology and Routing . . . 24
3.3 Maximum Inter-Packet Overhead . . . 25
3.4 Network Frequency . . . 26

3.5 Injection Control Strategy 28
3.6 Summary 32

4 Implementation **33**
4.1 Hardware Architecture 33
4.1.1 Microarchitectural Choices 33
4.1.2 System Architecture Overview 34
4.1.3 Router 36
4.1.4 Network Interface 38
4.2 RTL Verification Testbench 39
4.2.1 Overall Structure 40
4.2.2 Traffic Models and the Generator 41
4.2.3 Dynamic Frequency Controller 44
4.2.4 Other Components 45
4.3 Summary 46

5 Simulation and Hardware Result **47**
5.1 RTL Testbench Results 47
5.1.1 Functional Verification 48
5.1.2 Static-Frequency Link Utilization 49
5.1.3 Dynamic-Frequency Link Utilization 51
5.2 Hardware Result 54
5.2.1 Logic Synthesis 54
5.2.2 Physical Design 58
5.3 Summary 61

6 Conclusions and Future Work **63**
6.1 Conclusions 63
6.2 Limitations 64
6.3 Future work 64

References **65**

List of Figures

1.1 Overview of the thesis: Network-on-Chip Design for Specific Traffic Pattern. 4
1.2 Workflow of the thesis. 6

2.1 On-chip network topologies of mesh (left) and flattened butterfly (right), adapted from [15]. 11
2.2 Virtual-Channel router microarchitecture, adapted from [25]. . 15
2.3 Looped container in the Nostrum NoC platform, adapted from [28]. 16
2.4 Highway channel mechanism for improving TDM slot utilization, adapted from [30]. 16
2.5 Parallel probing algorithm for dynamic virtual-circuit setup, adapted from [33]. 17
2.6 Subchannel and subnetwork organization in MultiCS, adapted from [33]. 18
2.7 Port allocation mechanism discussed in CHIPPER, adapted from [40]. 18
2.8 MinBD router pipeline with a small side buffer, adapted from [41]. 19
2.9 One example of the hierarchical ring structure in HiRD, adapted from [42, 43]. 20

3.1 Example of integer multi-commodity flow. 23
3.2 Torus topology with an example path from node 0 to node 18. 25
3.3 Topology-level illustration of a collision between in-flight flits. 26
3.4 Space–time view of the in-flight flit collision scenario. 27
3.5 Required NoC frequency as a function of the number of flits per packet with a fixed per-packet overhead of 4 cycles 29
3.6 Comparison of the local controller and the centralized controller. 30

3.7 Tile injection is decoupled from the network by an input buffer in the network interface. . . . 31
3.8 Buffer occupancy under the local and centralized strategies, showing convergence to the same steady state. . . . 32

4.1 System architecture of the 5 × 5 torus NoC. . . . 35
4.2 Router microarchitecture. . . . 36
4.3 Arbiter microarchitecture. . . . 37
4.4 Network interface architecture. . . . 38
4.5 Finite state machine for local injection control. . . . 39
4.6 Architecture of the RTL verification testbench. . . . 40
4.7 Random permutation traffic generation procedure. . . . 42
4.8 Snapshot of generated random permutation traffic. . . . 43

5.1 QuestaSim waveform for a routed packet from node 0 to node 18. . . . 49
5.2 Dwell-time distribution for different frequencies and the corresponding packet lengths under fine-grained dynamic frequency scaling. . . . 52
5.3 Dwell-time distribution for different packet length levels under five-level quantized dynamic frequency scaling. . . . 53
5.4 Component-level area breakdown of a NoC node under different payload widths. . . . 55
5.5 Internal area proportion breakdown of the network interface and router under different payload widths. . . . 56
5.6 Area scaling with data path width for the synthesized NoC components. . . . 58
5.7 Floorplan of the 5 × 5 NoC with tile placeholders and adjacent NoC node regions. . . . 59
5.8 Post-layout placement-and-routing snapshot of the full 5 × 5 NoC. . . . 60

List of Tables

4.1 Flit format for one channel. 34

5.1 Measured link utilization across the random-traffic configurations. 54

5.2 Per-component linear fit $A = aW + b$ across $W \in \{16, 32, 64, 128\}$. 57

List of acronyms and abbreviations

ASIC	Application-Specific Integrated Circuit
AXI4	Advanced eXtensible Interface 4
DUT	Design Under Test
FFT	Fast Fourier Transform
FIFO	First-In, First-Out
FSM	Finite State Machine
GPU	Graphics Processing Unit
MCF	Multi-Commodity Flow
NI	Network Interface
NoC	Network-on-Chip
PLL	Phase-Locked Loop
RISC-V	Reduced Instruction Set Computer V
RTL	Register-Transfer Level
RX	Receive
SDM	Space-Division Multiplexing
TDM	Time-Division Multiplexing
TX	Transmit

Chapter 1

Introduction

1.1 Background

The continued scaling of process technology has enabled an increasing number of computational units to be integrated on a single chip. Modern many-core processors [1], Graphics Processing Units (GPUs)s [2], and domain-specific accelerators [3, 4] commonly integrate tens to hundreds of processing elements, each of which may generate inter-element communication traffic. As a result, the on-chip interconnect has become a major determinant of system performance, energy efficiency, and area.

Shared buses and monolithic crossbars are effective for small-scale multicore systems, but their bandwidth, wiring, arbitration, and area costs become increasingly difficult to scale as the number of cores grows. This limitation has motivated the use of on-chip interconnection networks, commonly referred to as Network-on-Chips (NoCs), as scalable communication fabrics for many-core systems [5, 6, 7]. For example, the Eyeriss accelerator chip, which integrates 168 processing elements for convolutional neural network workloads, uses a NoC to deliver data across the array of processing elements [3].

Synthetic traffic patterns are widely used to evaluate the behaviour of on-chip interconnection networks under controlled and repeatable communication scenarios. Unlike application traces, synthetic workloads isolate specific spatial communication structures, making it possible to stress particular aspects of a topology, routing algorithm, or flow-control mechanism [8, 9]. Common examples include uniform random, nearest-neighbour, transpose, tornado, hotspot, and permutation traffic [8]. These patterns cover both average and adversarial communication behaviours. This work focuses on

random permutation traffic, where each source is randomly assigned a unique destination. Under full offered load, this pattern provides a balanced stress case for network contention, because each node acts as exactly one source and one destination.

Beyond its role as a controlled synthetic benchmark, permutation traffic is also a useful abstraction for structured data movement in high-throughput computing workloads. For example, the inter-stage data movement of a radix-r Fast Fourier Transforms (FFTs) can be expressed as a sequence of structured stride permutations, and parallel-memory FFT processors are often organized to provide conflict-free access for such permutations [10]. In mixture-of-experts models, tokens are dispatched to different experts and the expert outputs are later combined, which introduces communication among the devices (or partitions) hosting the experts [11, 12]. In the idealized uniform case, an all-to-all exchange can be scheduled as a sequence of permutation rounds. More generally, admissible traffic matrices can be decomposed, after suitable normalization if necessary, into weighted permutation schedules using the Birkhoff-von Neumann construction employed in input-queued switch fabrics [13]. These examples show that permutation traffic is valuable not only as a controlled synthetic benchmark, but also as an abstraction for studying structured data movement in high-throughput systems.

1.2 Problem Definition

Designing a NoC for sustained permutation-style communication is non-trivial, even for a moderately sized network. A design that appears suitable at the architectural level may still encounter limitations when translated into Register-Transfer Levels (RTLs) and evaluated as hardware. Early design choices affect routing behaviour, timing, resource cost, and achievable throughput. The central problem is therefore to identify a NoC architecture and implementation that can support the target traffic while remaining practical as a hardware design.

In addition, many NoC studies focus on selected aspects of the design problem, such as topology selection, routing, or performance evaluation, while simplifying assumptions about the remaining parts of the design flow. This makes it difficult to assess how architectural assumptions translate into concrete RTL implementation, verification results, and physical-design costs. It is therefore useful to study a complete design point in which architectural analysis, RTL implementation, verification, and physical characterization are carried out under a consistent set of assumptions. Accordingly, this thesis

addresses the following research questions:

1. What NoC architecture can support the target permutation-style traffic, and what topology, routing, and injection-control constraints follow from this requirement?

2. How can the resulting architecture be implemented as a complete RTL design, including the microarchitectures of router and Network Interfaces (NIs), and supporting control logic?

3. Does the implemented design satisfy the target traffic requirements in simulation, and what area and timing costs does it incur after synthesis and preliminary physical design?

1.3 Purpose and Goals

The purpose of this thesis is to answer the research questions above by constructing and characterizing a complete network-on-chip design point for a 25-node system under permutation-style communication. The work is intended as an exploratory study. The architectural choices made along the way favour logical simplicity over optimality, where the two may conflict. Rather than optimizing any single aspect of the design in isolation, this thesis focuses on closing the loop from network-level analysis to RTL implementation, testbench-based verification, and physical characterization on a commercial 28-nm process.

The thesis pursues three goals, each corresponding to one of the research questions:

1. Develop and analyze a NoC architecture for the target permutation-style traffic. This includes formulating the routing problem, selecting the topology and routing strategy, and deriving the frequency, and injection-control requirements needed to sustain the target communication pattern.

2. Implement the proposed architecture as a complete RTL design. The implementation defines the router and NI that together form each node of the 25-node network. A testbench is also developed to generate traffic, observe network behaviour, and verify the delivered packets against a reference model.

3. Verify and characterize the implemented design. This includes functional verification under deterministic and random permutation traffic, measurement of link utilization under different operating conditions, and reporting of area and timing results after synthesis and preliminary physical design.

Together with the research questions in Section 1.2, the overview of this thesis is shown in Fig. 1.1.

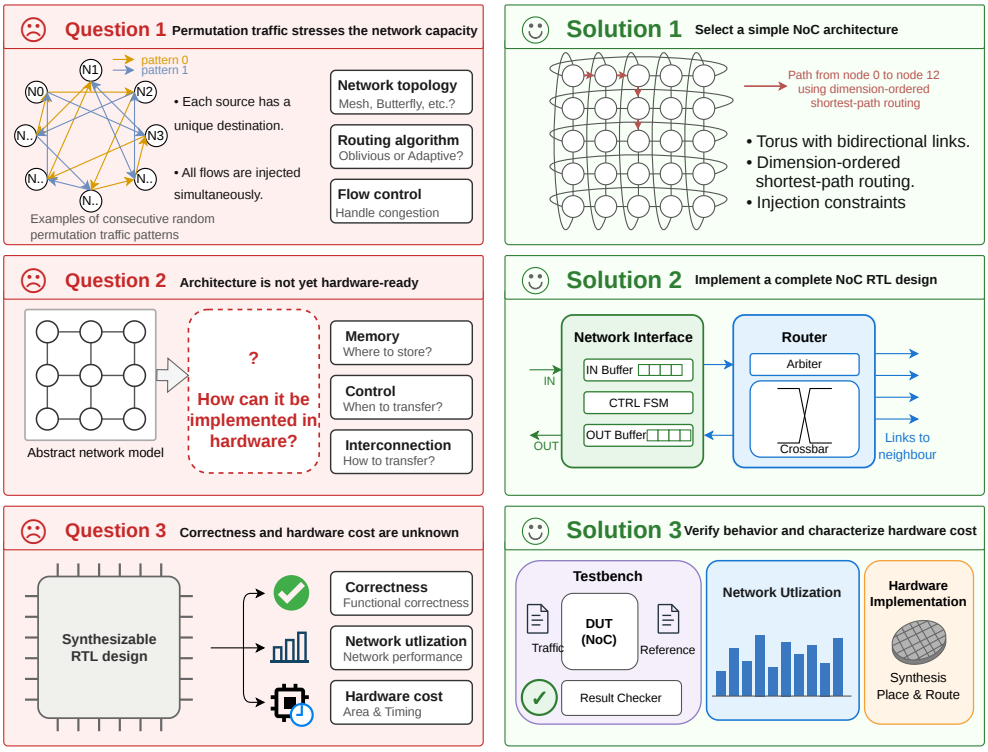


Figure 1.1: Overview of the thesis: Network-on-Chip Design for Specific Traffic Pattern.

1.4 Research Methodology

This thesis follows a constructive engineering methodology. Rather than comparing a large population of alternative NoC designs, the work constructs one complete design point and evaluates it consistently across analysis, implementation, simulation, and physical realization. The methodology is organized into four sequential stages, as shown in Fig. 1.2. Each stage

produces information that constrains or validates the next stage, while later stages provide feedback on the assumptions made earlier.

Stage 1 is architectural analysis. This stage formulates the traffic and routing problem, selects the topology and routing strategy to support the target permutation-style communication, and derives the timing and injection-control requirements needed by the implementation.

Stage 2 is RTL implementation. This stage translates the architectural requirements into a SystemVerilog design, including the router and NI required to build the complete network.

Stage 3 is simulation-based verification and measurement. This stage exercises the RTL using a self-checking testbench under adversarial and random permutation traffic. The purpose is to verify functional correctness and to measure dynamic behaviour such as link utilization under different operating conditions.

Stage 4 is hardware realization. This stage carries the verified RTL through logic synthesis and preliminary place-and-route on a commercial 28-nm process. The resulting area and timing reports are used to characterize the hardware cost and achievable operating point of the proposed design.

1.5 Ethics and Sustainability

This thesis does not involve human participants, personal data, or user-facing decision-making systems. The main ethical considerations are therefore related to research integrity, reproducibility, and responsible reporting of results. The design assumptions, traffic models, verification method, and physical-design limitations are stated explicitly so that the reported conclusions can be interpreted within their intended scope.

From a sustainability perspective, on-chip interconnect design is relevant because communication energy is a significant component of many-core and accelerator systems. Improving the efficiency and predictability of data movement can therefore contribute to more energy-efficient hardware. In addition, a transparent and reusable design flow can reduce unnecessary duplication of modeling, implementation, and evaluation effort. This work supports sustainable hardware research by providing a transparent and repeatable framework for studying NoC behaviour, routing, traffic patterns, and preliminary physical design characteristics.

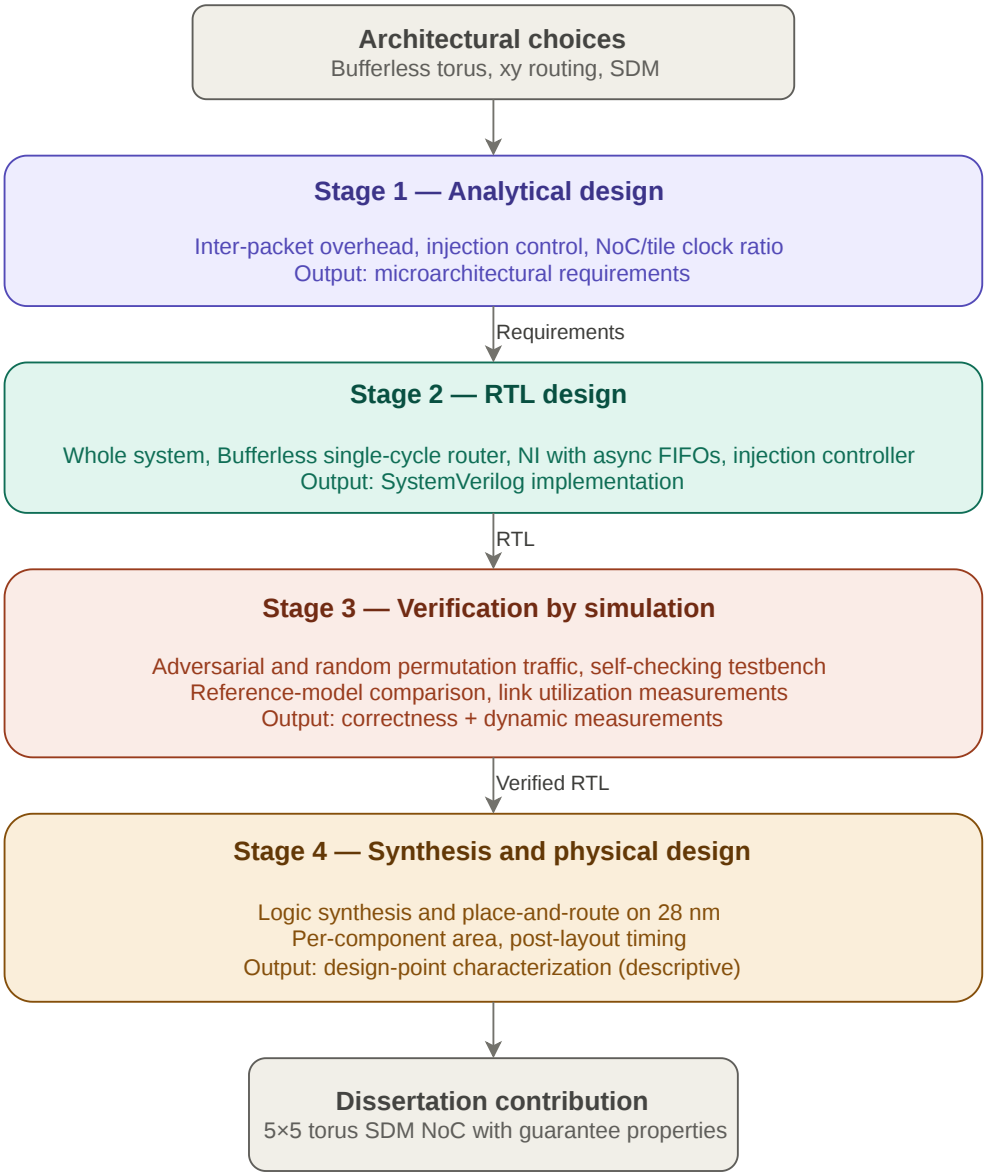


Figure 1.2: Workflow of the thesis.

1.6 Delimitations

The scope of this thesis is defined by three main delimitations. Together, they keep the work focused on the construction and characterization of a complete NoC design point under a clearly specified traffic model.

Targeted traffic model. The NoC is designed and evaluated for permutation-style traffic rather than for all possible on-chip communication patterns. The main evaluation cases are shift permutations and random permutation traffic. This scope is appropriate for studying sustained contention and link utilization under balanced source and destination demand, but the results should not be interpreted as a general performance prediction for workloads with strong locality, hotspot behaviour, multicast communication, or irregular application-dependent traffic.

Simulation-level dynamic frequency control. The thesis studies dynamic NoC frequency scaling as an architectural and simulation-level mechanism. The frequency controller used in the evaluation models the effect of changing the NoC operating frequency under different traffic conditions, but it is not implemented as a complete hardware frequency-control circuit. As a result, the reported results characterize the potential benefit of frequency scaling for this NoC design, rather than the full circuit-level cost of a practical dynamic frequency-control subsystem.

Preliminary physical realization. The RTL design is carried through logic synthesis and preliminary place-and-route on a commercial 28-nm process. This provides a concrete hardware characterization of the design point. However, the physical-design flow is not intended to be a sign-off-ready chip implementation and does not include final timing closure, full power sign-off, design-for-test insertion, or tapeout-level verification.

1.7 Structure of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 introduces the technical background needed for the rest of the thesis, including the basic components of a network-on-chip, common topologies, routing algorithms, switching mechanisms, and related work. Chapter 3 presents the network-level design and analysis. It formulates the routing problem, selects the topology and routing strategy, and derives the timing and injection-control requirements needed to support the target permutation-style traffic. Chapter 4 describes the RTL implementation of the proposed NoC, including

the router, NI, supporting control logic, and the self-checking verification testbench. Chapter 5 evaluates the implemented design through simulation and hardware realization. It reports functional verification results, link-utilization measurements under static and dynamic frequency operation, synthesis area, and preliminary post-layout timing on the 28-nm process. Chapter 6 concludes the thesis by summarizing the main findings, discussing the limitations of the work, and identifying directions for future research.

Chapter 2

Background

This chapter introduces the technical background required to follow the rest of the thesis. Section 2.2 presents the network-on-chip as a system architecture, the components that compose it, and the topologies in common use. Section 2.3 covers the two design decisions that determine how data actually moves through the network: the routing algorithm that selects a path, and the switching mode that determines how the path is held and how the flits traverse it. Section 2.4 surveys prior work in the areas most relevant to this thesis.

2.1 Motivation and Scope of Network-on-Chip

As process scaling has increased the number of computational units that can be integrated on a single chip, traditional shared buses and simple crossbars have become increasingly difficult to scale. A shared bus allows only limited concurrent communication and can become a serialization bottleneck when many components contend for access. A full crossbar can support multiple non-conflicting transfers, but its wiring and area cost grow as $O(N^2)$, making it unattractive for systems with tens or hundreds of cores [7]. The NoC addresses these limitations by replacing a monolithic interconnect with a distributed network of routers and links. This structure provides more scalable aggregate bandwidth and allows the physical organization of the interconnect to better match the floorplan of the chip.

NoC is now widely used in many-core processors [1], , GPUs [14], and domain-specific accelerators [3]. In such systems, many processing elements

may communicate concurrently, and the interconnect must support structured traffic patterns with predictable latency and bandwidth requirements. This thesis focuses on permutation-style traffic, in which each node acts as both a source and a destination and the source–destination mapping forms a bijection. This traffic model is simple enough for analytical study while still capturing an important class of structured many-to-many communication in regular on-chip networks.

2.2 Network-on-Chip Hardware Architecture

2.2.1 Fundamental components of Network-on-Chip

A NoC is typically composed of three main hardware elements: routers, links, and NIs [7].

The router is the active forwarding element of the network. It receives flits from its input ports, determines the appropriate output port for each flit, arbitrates among competing requests when multiple inputs target the same output, and forwards the selected flit onto the corresponding output link. Router designs differ in their pipeline depth, buffering strategy, arbitration logic, and the mechanisms used to multiplex multiple traffic flows over shared physical resources.

Links provide the physical communication paths between routers, or between a router and its attached NI. A bidirectional connection is commonly implemented as two unidirectional links carrying flits in opposite directions. The width of a link determines the nominal number of data bits that can be transferred per cycle, while the physical length of the link affects delay, timing closure, and the possible need for repeaters or pipeline registers.

The NI connects a processing element, referred to in this thesis as a tile, to the NoC router. On the transmit side, it accepts packets from the tile, formats them into flits, attaches the required routing or destination information, and injects the flits into the network. On the receive side, it collects incoming flits, reconstructs packets when necessary, and delivers them to the tile. When the tile and the network operate in different clock domains, the NI is also a natural location for clock-domain crossing logic.

2.2.2 Topology

The topology of a NoC is the abstract graph that specifies which routers are directly connected by links. It determines the diameter of the network, that is,

the maximum minimum hop count between any pair of nodes, and constrains both routing and physical floorplanning. Regular topologies are commonly used in NoC designs because they simplify router design, address assignment, and on-chip wiring.

The simplest two-dimensional topology is the *mesh*: an $N_x \times N_y$ grid in which each internal node is connected to its north, south, east, and west neighbours, while boundary nodes have fewer neighbours. Mesh networks are easy to lay out on a rectangular die and naturally support coordinate-based routing. Their main drawback is that the shortest-path diameter is $(N_x - 1) + (N_y - 1)$, so traffic between opposite corners requires a relatively large number of hops.

A *torus* extends a mesh by adding wrap-around links between the first and last nodes of each row and column. These wrap-around links reduce the shortest-path diameter to $\lfloor N_x/2 \rfloor + \lfloor N_y/2 \rfloor$ and, for normal torus dimensions, give all nodes the same degree. As a result, a torus removes the structural difference between central, edge, and corner nodes in a mesh. The cost is the additional wrap-around wiring, which can complicate physical design because these links may span long distances unless a folded layout is used.

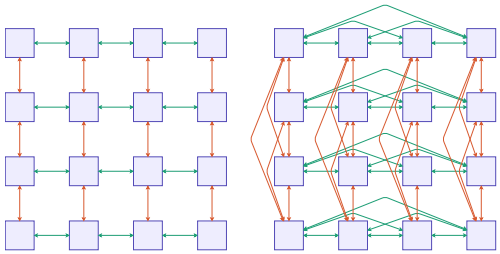


Figure 2.1: On-chip network topologies of mesh (left) and flattened butterfly (right), adapted from [15].

Higher-radix topologies, such as the flattened butterfly [16], shown in Fig. 2.1, and the dragonfly [17], reduce network diameter by increasing the number of ports per router and providing richer inter-router connectivity. Compared with traditional multistage Clos networks [18], these topologies can provide shorter paths and lower latency when high-radix routers are feasible. They have been studied mainly in the context of high-performance and large-scale interconnection networks. However, a larger router radix also increases crossbar size, wiring demand, and arbitration complexity, which can make such designs less attractive for dense, latency-sensitive single-die NoCs. Less regular topologies, such as random graphs or designer-tuned

irregular networks, can improve performance for specific traffic patterns, but they generally require more complex routing and floorplanning [19].

2.3 Routing and Switching

After the topology of a NoC has been fixed, several lower-level design choices determine how traffic is carried through the network. The *routing algorithm* selects the path, or equivalently the output direction at each router, for a packet or flit. The *switching mode* determines whether network resources are acquired and used packet by packet or reserved in advance for a connection. In addition, *resource-sharing* mechanisms determine how multiple logical flows share the same physical resources.

2.3.1 Routing Algorithms

A routing algorithm defines how a packet is forwarded from its source to its destination. Routing algorithms are commonly classified according to whether they depend on the current network state, and whether they are restricted to shortest paths.

Oblivious routing selects paths using only the source and destination addresses, without considering the current state of the network. The simplest and most widely used oblivious algorithm on a mesh or torus is *dimension-ordered routing*, often called XY routing. In XY routing, a packet is first forwarded along the X dimension until its X coordinate matches that of the destination, and is then forwarded along the Y dimension. This algorithm requires no routing tables, makes routing decisions in a few gates of logic, and is provably deadlock-free in mesh, because it avoids cycles in the channel-dependency graph [20, 8]. This routing method is a kind of deterministic algorithm, which means the path is always the same for the source–destination pair. Other oblivious algorithms may use randomized or non-minimal paths. For example, Valiant routing first sends a packet to an intermediate node and then forwards it to the final destination [21]. This can spread load more evenly under adversarial traffic patterns, but it increases the path length and therefore the latency.

Adaptive routing allows the routing decision to depend on the current network state, such as output-port availability or congestion. Adaptive algorithms can route around hot spots and improve throughput under non-uniform traffic, but it requires more complex routing logic. It also requires additional mechanisms to guarantee deadlock freedom, commonly through

virtual channels, turn-model restrictions, or escape paths [20, 22]. Since packets between the same source and destination may take different paths, adaptive routing may break in-order delivery unless additional ordering or reordering mechanisms are used.

A second classification is whether the selected route is minimal. *Minimal* routing always uses a shortest path between the source and destination, while *non-minimal* routing may take a longer path to avoid congestion or improve load balance. Many regular NoC designs use minimal routing as the default because it is simple and consumes fewer network resources per packet.

2.3.2 Switching Modes and Resource Sharing

Once the path has been selected, the switching mode determines how the network resources along that route are used. Two broad approaches are *packet switching* and *circuit switching*.

In *packet switching*, each packet is forwarded independently through the network. The packet may be divided into flits, and routers forward these flits according to the routing and flow-control mechanism. In store-and-forward switching, each intermediate router receives and stores the entire packet before forwarding it. This approach is simple but has high latency and buffering cost, making it uncommon in latency-sensitive NoCs. In *wormhole* switching, the head flit performs routing and the following flits proceed along the same path. Since routers do not need to store the entire packet, wormhole switching reduces buffering requirements and latency compared with store-and-forward switching [8]. Wormhole switching has become a dominant scheme in NoC designs [23] because it offers low latency and modest buffer requirements at the same time. Modern packet-switched NoC routers often combine wormhole switching with virtual-channel flow control to reduce head-of-line blocking and support deadlock avoidance.

In *circuit switching*, a path from source to destination is reserved before data transmission begins. After the setup phase, data can traverse the reserved path without performing a new routing decision at every router. This can provide predictable latency and contention-free transmission for the reserved connection, but it introduces setup overhead and holds network resources for the duration of the transfer. Circuit switching is well-suited to traffic patterns in which a single source–destination pair exchanges a large amount of data in a single burst.

Several mechanisms can be used to share physical links among multiple logical flows. *Time-Division Multiplexings (TDMs)* shares a link in time by

assigning different time slots to different flows, usually according to a static or periodically repeated schedule. This approach is often used in circuit-switched or guaranteed-throughput NoCs because it can provide predictable bandwidth. Virtual channel mechanisms divide a physical channel into multiple logical channels, each with separate buffering and flow-control state. It is commonly used in packet-switched wormhole routers to reduce blocking and to help enforce deadlock-free routing. *Space-Division Multiplexings (SDMs)* share link resources spatially by allocating different wires or sub-links to different flows.

2.4 Related Work

This section reviews prior work related to NoC. The discussion first covers buffered packet-switched NoCs, where virtual channels and router pipelines are commonly used. It then reviews circuit-switched NoCs based on TDM and SDM. Next, bufferless and minimally buffered NoCs are discussed as alternative designs that reduce router buffering. Finally, recent large-scale interconnect studies are briefly reviewed.

2.4.1 Packet Switched NoCs

Packet-switched NoCs transport messages as packets, which may be further divided into flits for flow control and router traversal. In a typical virtual-channel router, a head flit typically performs routing computation, virtual-channel allocation, switch allocation, and switch traversal, while body and tail flits only perform the last two stages [8]. Virtual channels are not required by packet switching itself, but they are widely used in wormhole packet-switched routers because they divide one physical channel into multiple logical channels with separate buffering and flow-control state. This helps reduce head-of-line blocking, support deadlock avoidance, and improve utilization under contention. The principle of virtual-channel flow control was established by Dally [24], and later low-latency virtual-channel router designs optimized the router pipeline for on-chip networks [25], whose microarchitecture is shown in Fig. 2.2. Recent work continues to reduce the cost and improve the scalability of packet switched NoCs. VVQ virtualizes virtual channels to reduce the number of physical virtual channels required for protocol deadlock avoidance [26]. Instead of assigning separate physical virtual channels to every protocol class, VVQ uses a more flexible virtual queue structure that allows packets to bypass queue blocking when protocol deadlock could occur.

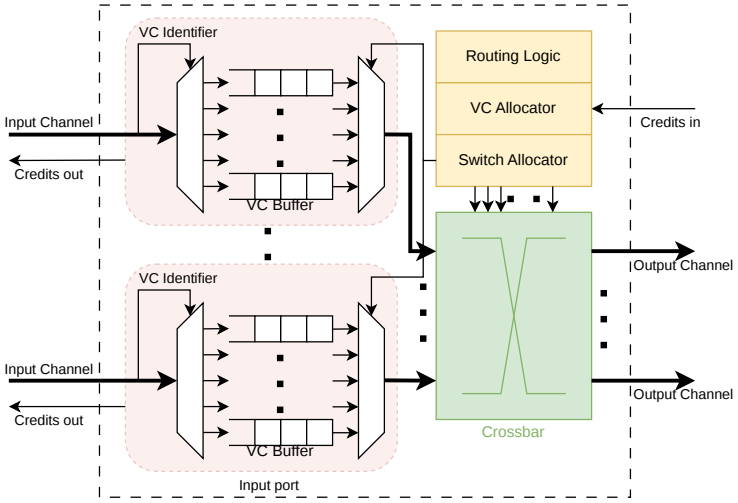


Figure 2.2: Virtual-Channel router microarchitecture, adapted from [25].

FlooNoC targets high-bandwidth accelerator traffic, using wide physical links and end-to-end Advanced eXtensible Interface 4s (AXI4s) support [27]. It is motivated by the bulk data transfers common in domain-specific accelerators, where the traffic differs from the small cache-line transfers often assumed in traditional cache-coherent multiprocessors.

2.4.2 Circuit Switched TDM NoCs

Circuit-switched NoCs reserve communication resources before data transfer. In TDM NoCs, link and router resources are divided into recurring time slots, and a connection is assigned compatible slots along all links of its path. This approach can provide guaranteed bandwidth and predictable latency when the schedule is feasible, but it requires slot allocation and may leave resources underutilized when the reserved schedule does not match the offered traffic.

Based on the Nostrum NoC platform, Millberg *et al.* proposed temporally disjoint networks and looped containers to provide guaranteed bandwidth [28]. In this design, containers circulate along prearranged closed paths, and a packet can enter a container only when it is empty. Contention freedom is achieved because the temporal allocation prevents two containers from requiring the same network resource at the same time. Æthereal NoC provides another influential example of guaranteed-service NoC design, supporting both guaranteed-throughput connections and best-effort traffic [29].

To improve slot utilization, Liu *et al.* proposed Highway NoC [30].

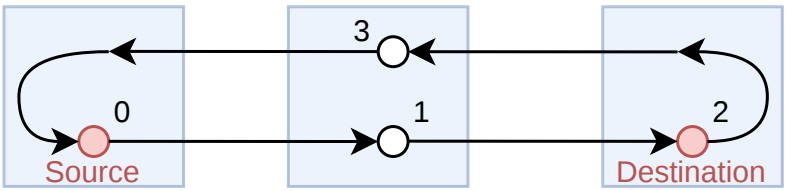


Figure 2.3: Looped container in the Nostrum NoC platform, adapted from [28].

Highway channels are added to a TDM NoC so that flits can temporarily wait and later use slots that would otherwise remain unused. This improves utilization when strict consecutive-slot allocation would leave gaps in the schedule.

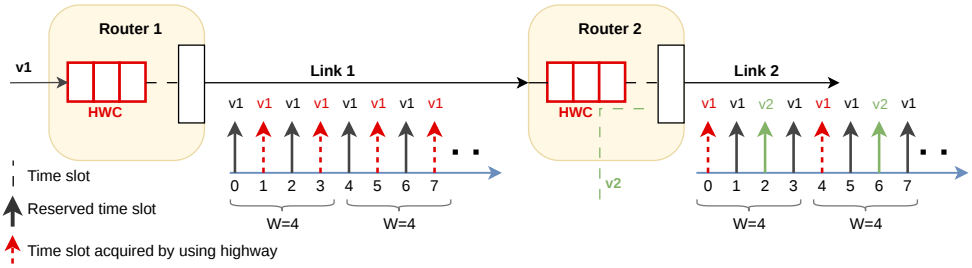
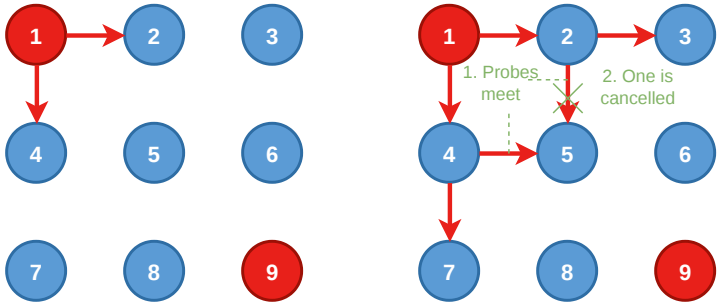


Figure 2.4: Highway channel mechanism for improving TDM slot utilization, adapted from [30].

Virtual-circuit allocation has also been studied for both static and dynamic settings. Static allocation can be based on logical networks, where compatible slot sequences are grouped and assigned to different virtual circuits to avoid conflicts [31]. Dynamic connection setup allows virtual circuits to be established at run time. Parallel probing methods propagate setup probes through the network to search for an available path and release redundant reservations when probes meet [32, 33].

More recent comparisons between TDM and best-effort NoCs highlight the main trade-off between predictability and implementation cost [34]. Best-effort NoCs are work-conserving and can provide lower zero-load latency, but they often require virtual-channel buffers, allocators, arbiters, and credit-based flow control. TDM NoCs avoid many of these structures but require a reservation schedule and can be less flexible under dynamic traffic.



(a) In each node a probe may double. (b) When two probes meet, one is cancelled.

Figure 2.5: Parallel probing algorithm for dynamic virtual-circuit setup, adapted from [33].

2.4.3 Circuit Switched SDM NoCs

SDM provides another way to reserve network resources. Instead of assigning different connections to different time slots, SDM divides resources spatially, for example by partitioning the link width into subchannels or by providing multiple physically separate channels. Leroy *et al.* proposed SDM-based circuit switching for NoCs and showed that, for a limited number of concurrent circuits, SDM can reduce area, energy, and setup overhead compared with TDM virtual circuits [35]. A later work presented router architecture and end-to-end implementation details for this approach [36].

MultiCS extends SDM by combining multiple channels and multiple subnetworks [33]. Subchannels divide a physical link into independent parts, while subnetworks separate resources into different layers that do not interfere with each other. This increases connection flexibility and allows multiple circuits to coexist spatially.

2.4.4 Bufferless and Minimally Buffered NoCs

Another line of work reduces or removes in-router buffering. Bufferless NoCs can reduce area and power by eliminating most input buffers, but output-port contention must then be handled without simply storing blocked flits inside the router. Moscibroda and Mutlu made the case for bufferless routing in on-chip networks, arguing that the removal of buffers can provide significant area and power savings at the cost of higher deflection and possible performance degradation under heavy load [37].

SCARAB is a single-cycle bufferless router that uses separate allocation,

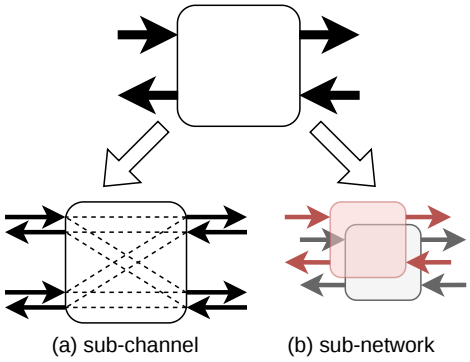


Figure 2.6: Subchannel and subnetwork organization in MultiCS, adapted from [33].

data, and NACK networks [38]. An allocation header is sent ahead of the data flits to configure the switch path. If allocation fails, the packet is dropped and a NACK is returned to the source, which triggers retransmission.

BLESS uses deflection routing instead of retransmission [39]. When multiple flits request the same output port, one flit obtains the productive output while the others are assigned alternative outputs and continue moving through the network. BLESS studies several flit-ranking policies, including closest-first and oldest-first, to improve progress and avoid livelock. It also proposes flit-level and wormhole-style variants. Deflection routing removes router buffers, but it can increase path length, introduce out-of-order arrival, and require endpoint reassembly.

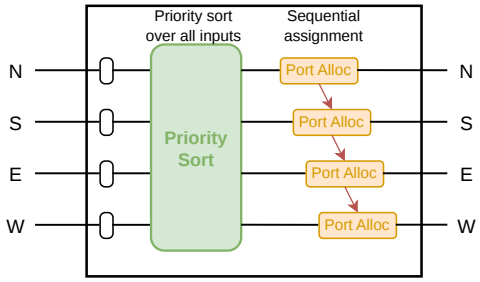


Figure 2.7: Port allocation mechanism discussed in CHIPPER, adapted from [40].

CHIPPER reduces the implementation complexity of BLESS by replacing complex age-based prioritization with the Golden Packet mechanism and by

using a permutation network for simplified arbitration [40]. The Golden Packet receives the highest priority and is guaranteed to make progress, which avoids the need for global age comparison. CHIPPER also proposes Retransmit-Once flow control to handle correctness issues related to packet delivery and reassembly.

MinBD combines bufferless deflection routing with a small side buffer [41]. When a flit would otherwise be deflected, the router may place it into the side buffer and reinsert it later into the pipeline. This reduces unnecessary deflections while retaining much of the area and power benefit of bufferless routing. MinBD also extends the priority mechanism with an additional silver priority level to reduce deflection and improve throughput.

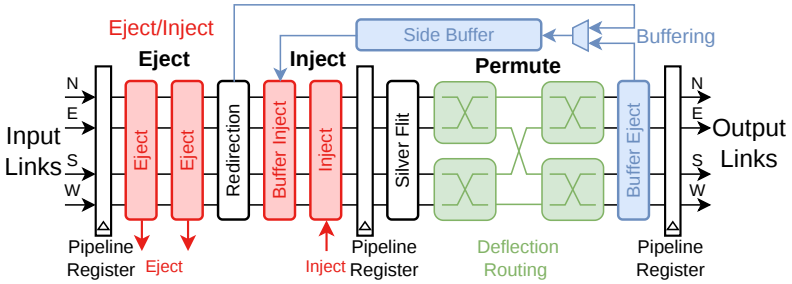


Figure 2.8: MinBD router pipeline with a small side buffer, adapted from [41].

HiRD applies deflection-based ideas to a hierarchical ring interconnect [42, 43]. The lowest level consists of local rings with bufferless node routers, while higher levels use bridge routers to connect rings. When a bridge buffer is unavailable, a flit can continue circulating in the ring and try again later. This avoids dropping the flit while keeping the local ring simple.

A recent multi-ring NoC for a heterogeneous chiplet system further develops ring-based interconnect organization [44]. The design uses half-ring and full-ring structures for different node counts and latency requirements, and it introduces different bridge structures for intra-die and inter-die communication.

FastPass is a more recent example of reducing buffering and virtual-channel overhead through bypassing [45]. It provides predefined nonoverlapping bypass paths so that selected packets can move across multiple hops without being blocked by intermediate routers. This improves progress and reduces the dependence on conventional virtual channels for both performance and deadlock avoidance.

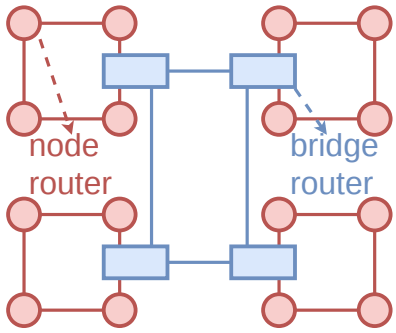


Figure 2.9: One example of the hierarchical ring structure in HiRD, adapted from [42, 43].

2.4.5 Recent Large-Scale Interconnect Studies

Recent systems show that NoC and interconnect design remain active research topics in many-core processors, GPUs, accelerators, and wafer-scale systems. HammerBlade is a recent open-source Reduced Instruction Set Computer Vs (RISC-Vs) manycore architecture that has been validated in silicon with a 2048-core Application-Specific Integrated Circuits (ASICs) implementation [1]. Its scale demonstrates the importance of scalable tiled organization and on-chip communication in modern many-core research systems. Real GPU NoC characterization work studies several generations of NVIDIA GPUs and reports that NoC latency and bandwidth can be nonuniform across physical locations and architectural partitions [46]. These observations show that practical NoC behaviour can depend strongly on physical organization as well as logical topology.

Waferscale Network Switches studies the use of wafer-scale integration for building high-radix network switches [47]. Although the target is a network switch rather than a conventional core-to-core NoC, the work is relevant to interconnect design because it shows how scaling is limited by physical factors such as I/O bandwidth, power delivery, cooling, and the wafer-scale substrate. It provides a recent example of how interconnect architecture, packaging technology, and physical constraints must be considered together.

Chapter 3

Network Design and Analysis

This chapter presents the design of the proposed NoC and analyses its key properties under an idealized model. The chapter is organized as follows. Section 3.1 formalizes the routing problem as a Multi-Commodity Flows (MCFs) feasibility problem, abstracting away hardware-specific details such as latency and pipelining. Section 3.2 introduces the chosen topology and routing scheme. The remaining sections address three design issues that arise once the abstract model is grounded in hardware: the inter-packet overhead induced by pipelined transport (Section 3.3), the network operating frequency required to sustain lossless transfer under the assumed traffic and injection-control model (Section 3.4), and the choice between centralized and local injection control (Section 3.5).

3.1 Multi-Commodity Flow Formulation

MCF formulations a common formulation for capacitated network routing problems, where multiple source–destination demands share limited link resources [48]. It has also been used in NoC design and mapping [49, 50]. To help understand the traffic problem, the traffic and link-load behavior of the network is abstracted using the MCF model. Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. In the NoC case, a vertex corresponds to a network node or router, and an edge corresponds to a physical communication link.

A commodity represents one source–destination communication demand. Let K denote the set of commodities. For each commodity $k \in K$, let $s_k \in V$ be the source vertex, $t_k \in V$ be the destination vertex, and d_k be the amount of demand. Since each edge is undirected, a commodity may use the edge in

either direction. For an edge $\{i,j\} \in E$, directional flow variables f_{ij}^k and f_{ji}^k are introduced to describe the flow of commodity k from i to j and from j to i, respectively. The capacity of edge $\{i,j\}$ is denoted by c_{ij}.

In a general MCF formulation, each commodity must satisfy flow conservation at every vertex, while the total amount of flow using each edge must not exceed the edge capacity:

$$\sum_{u:\{u,v\}\in E} f_{uv}^k - \sum_{w:\{v,w\}\in E} f_{vw}^k = \begin{cases} -d_k & v = s_k \\ d_k & v = t_k \\ 0 & \text{otherwise} \end{cases} \qquad \forall v \in V,\, k \in K, \tag{3.1}$$

$$\sum_{k\in K} \left(f_{ij}^k + f_{ji}^k\right) \le c_{ij} \quad \forall \{i,j\} \in E, \tag{3.2}$$

$$f_{ij}^k, f_{ji}^k \ge 0. \tag{3.3}$$

The formulation above allows fractional flow, meaning that a commodity may be split across multiple paths. This is useful in many mathematical flow problems, but in this work, both the link capacity and the commodity flow are treated as integer quantities. Moreover, each commodity is unsplittable: one demand must be routed along one path instead of being divided among multiple paths. Therefore, for the unit-load case considered here, the flow variables are restricted to binary values:

$$f_{ij}^k, f_{ji}^k \in \{0,1\}. \tag{3.4}$$

This integer-flow interpretation provides a compact way to describe link contention. Fig. 3.1 shows a small example in which each edge has a capacity of two. Each colored arrow represents one unit of commodity flow. A flow assignment is feasible only when every link satisfies its capacity constraint, i.e., the total number of flows using it in both directions does not exceed its capacity. For example, a link carrying two units of flow satisfies the constraint, whereas a link carrying three units of flow violates it, making the assignment infeasible.

A traffic instance can then be described by specifying the commodity set K. In the permutation-traffic case considered in this analysis, the traffic pattern is represented by a permutation $\pi : V \to V$. Each vertex is associated with one unit-load commodity whose source is the vertex itself and whose destination is determined by the permutation:

$$s_k = v, \qquad t_k = \pi(v), \qquad d_k = 1. \tag{3.5}$$

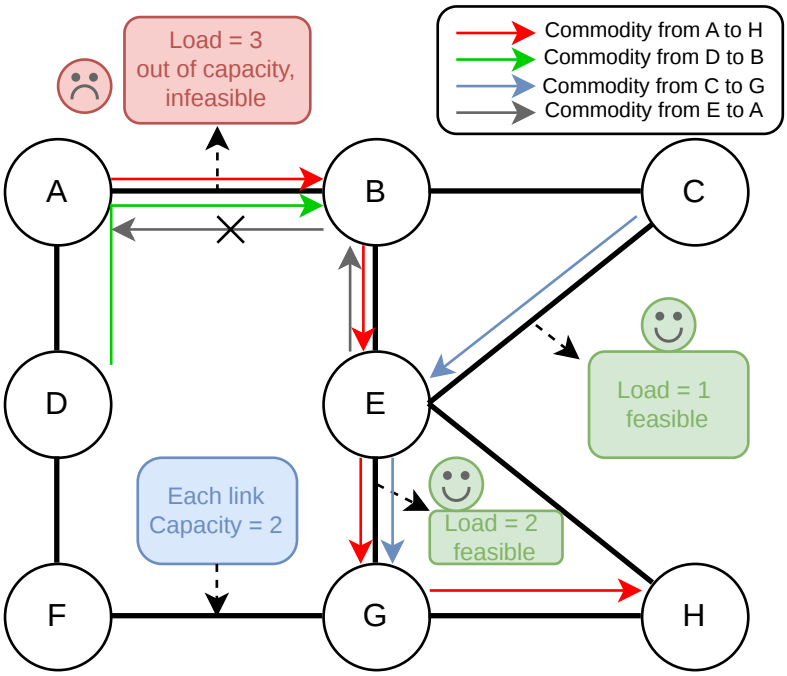


Figure 3.1: Example of integer multi-commodity flow.

Thus, each vertex acts as the source of one commodity and the destination of one commodity.

For a given traffic instance induced by π, the corresponding integer MCF constraints can be written as

$$\sum_{u:\{u,v\}\in E} f_{uv}^k - \sum_{w:\{v,w\}\in E} f_{vw}^k = \begin{cases} -1 & v = s_k \\ 1 & v = t_k \\ 0 & \text{otherwise} \end{cases} \qquad \forall v \in V,\, k \in K(\pi), \tag{3.6}$$

$$\sum_{k\in K(\pi)} \left(f_{ij}^k + f_{ji}^k\right) \le c_{ij} \quad \forall \{i,j\} \in E, \tag{3.7}$$

$$f_{ij}^k, f_{ji}^k \in \{0,1\} \quad \forall \{i,j\} \in E,\, k \in K(\pi). \tag{3.8}$$

These constraints capture the mathematical essence of the link-load feasibility condition: every commodity must form a valid source–destination path, and the aggregate load on each link must remain within the assigned capacity.

3.2 Topology and Routing

Chapter 2 introduced a broad design space for NoC. Since this thesis is exploratory rather than comparative, the design favours simpler mechanisms when they are sufficient for the target traffic model. This section describes the resulting topology and routing choices.

The proposed network adopts a 5×5 two-dimensional torus topology with bidirectional links, as shown in Fig. 3.2. The torus is chosen over a mesh for three main reasons. First, the wrap-around links reduce the worst-case shortest-path distance. For an $N \times N$ mesh, the maximum Manhattan distance is $2(N-1)$, whereas for an $N \times N$ torus it is $2\lfloor N/2 \rfloor$. In the 5×5 case, this reduces the worst-case distance from 8 hops to 4 hops. Second, the torus is structurally symmetric: every node has the same degree and the same relative position in the topology. This avoids the boundary effects present in a mesh, where corner and edge nodes have fewer neighbours than interior nodes. Third, the wrap-around links provide more path diversity: between any pair of nodes, this topology provide more available paths, which could relaxes congestion under the same traffic, although the deterministic routing strategy used in this thesis selects only one of these possible paths for each source–destination pair.

Among the routing strategies surveyed in the Chapter 2, adaptive routing offers higher resilience under non-uniform traffic, but at the cost of additional logic and the potential complications for in-order delivery guarantees. This thesis instead adopts a deterministic dimension-ordered shortest-path routing strategy, which is the simple to implement and analyze; in particular, it requires no routing tables and minimizes the per-hop routing decision logic. Under this scheme, a packet first traverses the row dimension until its column index matches that of the destination, then traverses the column dimension. The wrap-around links are used whenever they yield a shorter path.

As an example, consider node 0 sending a flit to node 18 in Fig. 2.1. The flit is first sent to node 4 through the horizontal wrap-around link, then forwarded to node 3, then to node 23 through the vertical wrap-around link, and finally to node 18. The resulting torus path requires only four hops, whereas the corresponding shortest mesh path would require six hops.

The remainder of this chapter examines the consequences of grounding this idealized model in real hardware, where flits traverse links over multiple clock cycles.

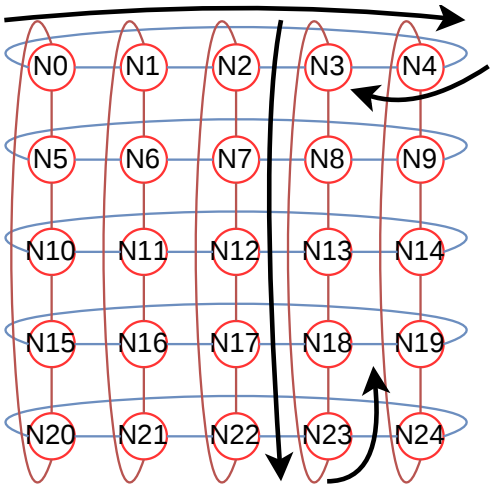


Figure 3.2: Torus topology with an example path from node 0 to node 18.

3.3 Maximum Inter-Packet Overhead

In the pipelined wormhole-style transport used in this design, assuming the router is a single-cycle-per-hop design without buffers, which simplifies both the analysis and the subsequent implementation, a flit injected at the source reaches the destination after a number of cycles equal to the Manhattan distance of the routing path. This finite per-hop latency introduces a timing constraint between consecutive packets sharing a destination, which the abstract MCF model of Section 3.1 does not capture.

Consider two consecutive packets targeting the same destination. Let the last flit of the previous packet be injected at cycle T, and let the distance from its source to the destination be d_{prev}. This last flit reaches the destination at cycle $T + d_{\text{prev}}$. A new packet from another source, at distance d_{new} from the same destination, can inject its first flit only after enough delay to ensure that it does not arrive before the previous packet has completed at the destination. If the new packet is injected after Δ cycles, its first flit arrives at cycle $T+\Delta+d_{\text{new}}$. To avoid destination-port contention or packet interleaving,

$$T + \Delta + d_{\text{new}} \geq T + d_{\text{prev}} + 1. \tag{3.9}$$

Equivalently,

$$\Delta \geq d_{\text{prev}} - d_{\text{new}} + 1. \tag{3.10}$$

If W denotes the number of idle waiting cycles between the two injections,

then $\Delta = W + 1$, and therefore

$$W \geq \max(0, d_{\text{prev}} - d_{\text{new}}). \tag{3.11}$$

The worst case on the 5×5 torus occurs when the previous packet travels the maximum distance and the new packet travels the minimum non-zero distance. This scenario is illustrated in Fig. 3.3, with an equivalent depiction given in Fig. 3.4. Source s_0 at distance $d_0 = 1$ is ready to inject its first flit of a new packet bound for dst, while source s_1 at distance $d_1 = 4$ from the destination has just injected the last flit of a packet bound for the same destination. Without sufficient injection delay, the first flit from s_0 would arrive at dst before the last flit from s_1, causing contention at the destination port

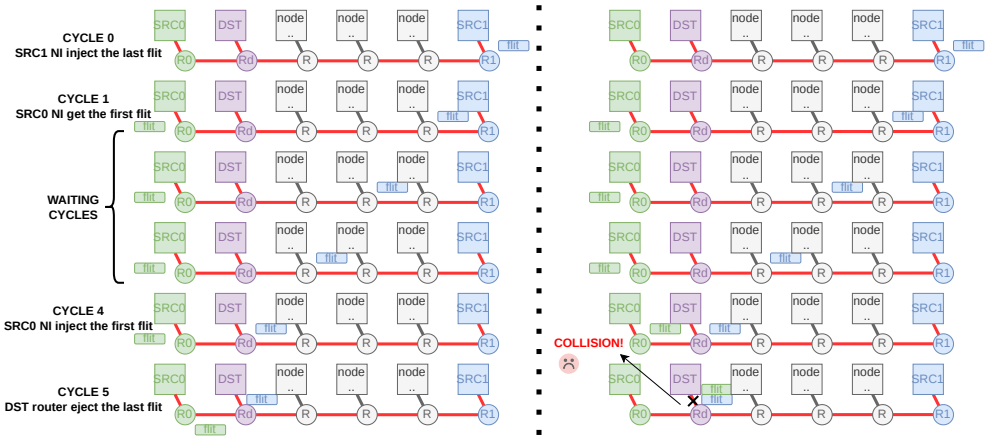


Figure 3.3: Topology-level illustration of a collision between in-flight flits.

This gives a maximum waiting time of $W_{\text{max}} = 3$ idle cycles. Including one additional cycle of NI overhead, the maximum per-packet overhead of this design is

$$T_{\text{overhead}} = 4 \text{ cycles}. \tag{3.12}$$

3.4 Network Frequency

Consider a tile that injects flits at a fixed frequency f_{tile}, with no knowledge of the network's internal state. The heaviest load the network can be subjected to is then a tile injecting continuously, one flit every tile cycle with no gaps. To sustain lossless transfer under the assumed permutation traffic, routing strategy, and injection-control policy, the NoC must drain flits at least as fast

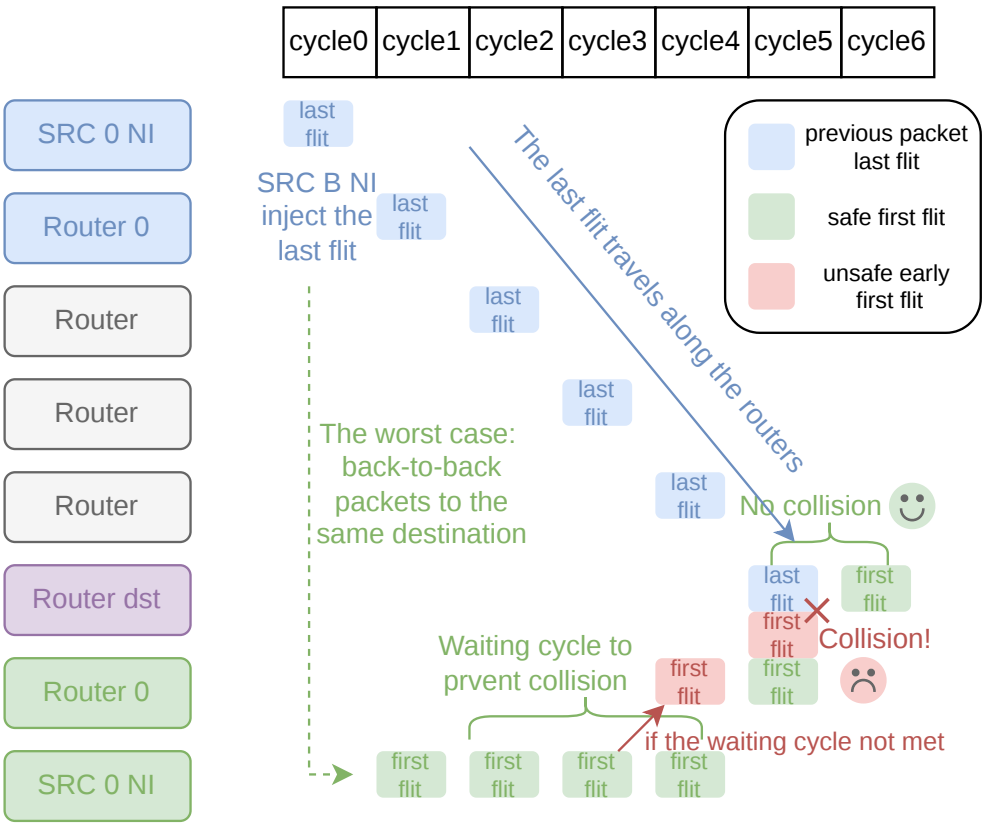


Figure 3.4: Space–time view of the in-flight flit collision scenario.

as the tile produces them. Equating the time for the tile to inject n_{flit} flits with the time for the NoC to transport those flits plus the per-packet overhead gives

$$\frac{n_{\text{flit}}}{f_{\text{tile}}} = \frac{n_{\text{flit}} + n_{\text{overhead}}}{f_{\text{NoC}}}, \tag{3.13}$$

which rearranges to

$$f_{\text{NoC}} = f_{\text{tile}} \cdot \frac{n_{\text{flit}} + n_{\text{overhead}}}{n_{\text{flit}}}. \tag{3.14}$$

The required frequency ratio depends on two quantities of different origin. The packet length n_{flit} is a workload parameter, fixed by the application rather than by the network. The per-packet overhead n_{overhead}, by contrast, is a property of the NoC itself, arising from the routing distance as derived in the previous section. To guarantee correctness under all assumed conditions, the design must size f_{NoC} for the worst case: the minimum packet length $n_{\text{flit}} = 8$ combined with the maximum overhead $n_{\text{overhead}} = 4$. This yields $f_{\text{NoC}} = 1.5 f_{\text{tile}}$.

This 1.5× ratio is conservative: the network does not carry minimum-length packets on every flow at all times, nor do all flows simultaneously traverse the worst-case routing distance. Whenever a packet is longer than the minimum or follows a shorter path, the actual required ratio is lower, and a value as low as 1.0–1.2× may suffice under those lighter conditions. Running the NoC at the static worst-case frequency therefore over-provisions network bandwidth for many transfers, as shown in Fig. 3.5. Section 5.1.3 (in a later chapter) evaluates a dynamic-frequency scheme that adapts f_{NoC} to the actual packet length and routing distance to recover this slack.

3.5 Injection Control Strategy

The waiting requirement $W \geq \max(0, d_{\text{prev}} - d_{\text{new}})$ derived in Section 3.3 depends on d_{prev}, which is the distance travelled by the previous packet targeting the same destination. However, this quantity is not directly observable at the new source, and two design choices address the missing information: a *local* controller per node that always assumes the worst case, and a *centralised* controller that has visibility over all in-flight packets. The remainder of this section quantifies the latency difference between the two and discusses their hardware cost.

Under the local strategy, the new source always assumes $d_{\text{prev}} = d_{\max} = 4$,

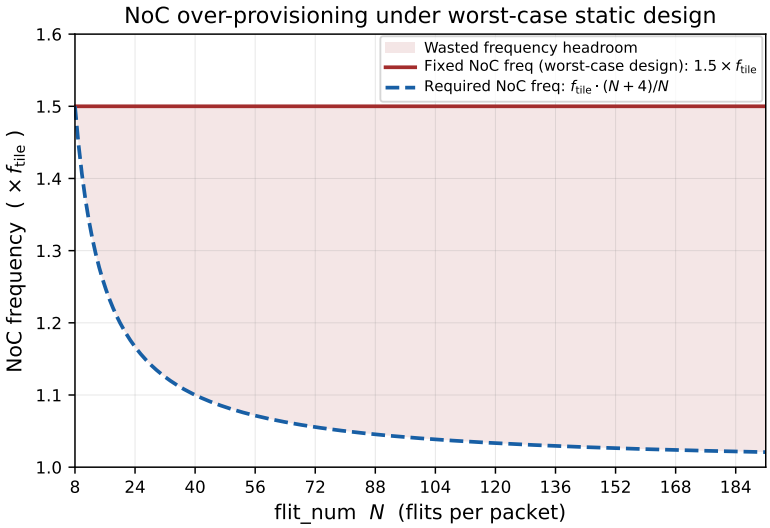


Figure 3.5: Required NoC frequency as a function of the number of flits per packet with a fixed per-packet overhead of 4 cycles

which represent the worst case, and waits $W_{\mathrm{L}}(d_{\mathrm{new}}) = 4 - d_{\mathrm{new}}$ cycles. Under the centralized strategy, there is a globally-aware controller supplies the actual d_{prev}, so the required waiting is the exact $W_{\mathrm{C}}(d_{\mathrm{prev}}, d_{\mathrm{new}}) = \max(0,\, d_{\mathrm{prev}} - d_{\mathrm{new}})$ of Section 3.3.

To compare this two strategies, consider the destination chosen uniformly at random among the 24 non-self nodes of the 5×5 torus. Direct enumeration yields the Manhattan distance distribution

$$\Pr[d = 1] = \tfrac{1}{6}, \quad \Pr[d = 2] = \tfrac{1}{3}, \quad \Pr[d = 3] = \tfrac{1}{3}, \quad \Pr[d = 4] = \tfrac{1}{6}. \tag{3.15}$$

The expected per-packet waiting time under the local strategy is

$$\mathbb{E}[W_{\mathrm{L}}] = \sum_{d_{\mathrm{new}}=1}^{4} \Pr[d_{\mathrm{new}}]\,(4 - d_{\mathrm{new}}) = \tfrac{1}{6}\cdot 3 + \tfrac{1}{3}\cdot 2 + \tfrac{1}{3}\cdot 1 + \tfrac{1}{6}\cdot 0 = \tfrac{3}{2} \text{ cycles}. \tag{3.16}$$

Treating d_{prev} and d_{new} as independent samples from (3.15), the centralized

counterpart is

$$\mathbb{E}[W_{\mathrm{C}}] = \sum_{d_{\mathrm{prev}}=1}^{4} \sum_{d_{\mathrm{new}}=1}^{4} \Pr[d_{\mathrm{prev}}] \Pr[d_{\mathrm{new}}] \max(0, d_{\mathrm{prev}} - d_{\mathrm{new}}) = \frac{19}{36} \approx 0.528 \text{ cycles.} \tag{3.17}$$

The mean per-packet head-latency penalty of the local strategy relative to the centralized one is therefore

$$\mathbb{E}[W_{\mathrm{L}}] - \mathbb{E}[W_{\mathrm{C}}] = \frac{3}{2} - \frac{19}{36} = \frac{35}{36} \approx 0.97 \text{ cycles per packet,} \tag{3.18}$$

which is an upper bound on the latency benefit obtainable from global awareness under this simplified independent-distance model.

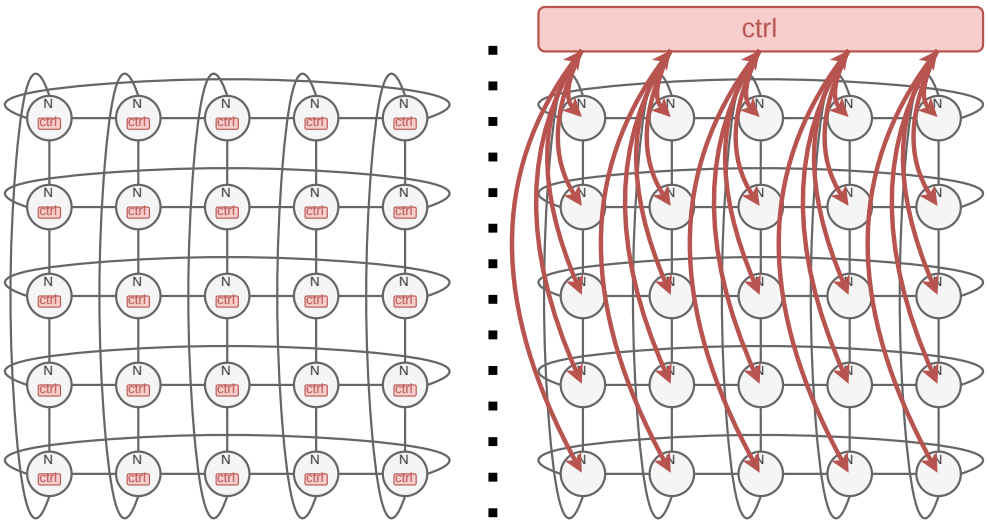


Figure 3.6: Comparison of the local controller and the centralized controller.

The two strategies differ substantially in implementation cost, as shown in Fig. 3.6. The local controller requires only a small Finite State Machines (FSMs) in each NI and applies the constant offset $4 - d_new$, where d_new is computed from the locally known destination address. No inter-node coordination is required, and the controller scales as $O(1)$ per node. The centralized scheduler, in contrast, must maintain the in-flight state of all $N = 25$ nodes and dispatch timing decisions back to every source. Realizing this requires either dedicated point-to-point signaling between the global controller and every node or an auxiliary control network spanning the chip. Both options introduce significant wiring overhead, place additional pressure on

timing closure, and make the controller a global synchronization point that constrains the achievable clock frequency. The total control state grows as $O(N)$.

Furthermore, the latency gap of (3.18) does not translate into a sustained throughput advantage, because injection is decoupled from the network by an input buffer in the NI, as shown in Fig. 3.7. A tile writes flits into this buffer at f_{tile}, and the NoC drains them at $f_{\text{NoC}} = 1.5 f_{\text{tile}}$ (Section 3.4). Because the NoC consumes flits faster than the tile produces them, the waiting cycles imposed by either control strategy only cause the buffer to fill temporarily; the surplus drain rate then closes this backlog within a bounded number of packets. Once the backlog is cleared, the system is *input-bound*: the buffer no longer accumulates, and the aggregate throughput equals the tile injection rate regardless of the control strategy. The centralized controller, whose per-packet waiting is shorter on average, reaches this input-bound regime earlier, while the local controller converges to the same steady-state throughput after a slightly longer transient.

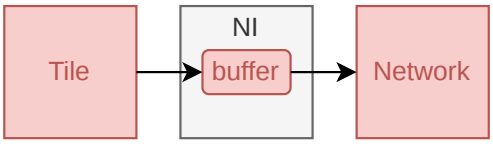


Figure 3.7: Tile injection is decoupled from the network by an input buffer in the network interface.

The two strategies therefore differ only during the warm-up transient, not in steady state, as illustrated by the buffer-occupancy traces in Fig. 3.8. In this example the local controller waits 3 cycles before its first injection while the centralized controller waits none. Under the local controller, this conservative waiting causes the buffer to fill (the *buffer fill* phase); the faster NoC then works this backlog down (the *catch-up drain* phase) until occupancy settles into a low, stable pattern (*steady state*). The centralized controller, injecting without waiting, reaches steady state almost immediately. After the first few packets, however, the two converge to identical behaviour. The local strategy thus pays a one-time transient cost on the order of (3.18) per packet during warm-up, which is negligible compared with the hardware cost of providing global awareness. The local controller is adopted in the proposed design.

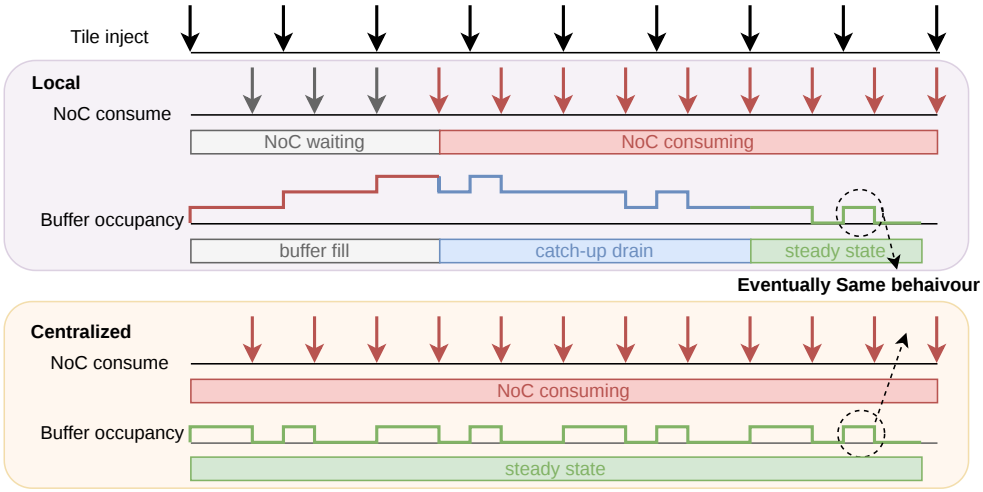


Figure 3.8: Buffer occupancy under the local and centralized strategies, showing convergence to the same steady state.

3.6 Summary

This chapter presented the design rationale and analytical basis of the proposed NoC. The routing problem was first formulated as an integer multi-commodity flow feasibility problem, providing an abstract way to reason about permutation traffic and link-capacity constraints. A 5×5 bidirectional torus with deterministic dimension-ordered shortest-path routing was adopted for its symmetry, reduced worst-case distance, and simple hardware implementation. The chapter then examined the timing effects introduced by pipelined flit transport, showing that inter-packet spacing is required to avoid destination-port contention. From the worst-case packet length and routing overhead, the required NoC frequency was derived as $1.5 \times f_{\text{tile}}$, while also motivating later dynamic-frequency optimization. Finally, local and centralized injection-control strategies were compared. Although centralized control can reduce transient waiting time, the local controller achieves the same steady-state throughput with much lower hardware and wiring cost, and is therefore selected for the proposed design.

Chapter 4

Implementation

This chapter describes the implementation of the proposed Network-on-Chip. Section 4.1 presents the hardware architecture, covering the top-level system, the bufferless router, and the NI. Section 4.2 describes the SystemVerilog testbench used to verify the design under synthetic permutation traffic and to collect the performance metrics reported in the next chapter.

4.1 Hardware Architecture

4.1.1 Microarchitectural Choices

Following the same design principle introduced in Section 3.2, the implementation favours simpler mechanisms when they provide sufficient functionality for the target traffic model. On this basis this implementation adopts two main microarchitectural choices:

- **SDM mechanism.** Among the possible mechanisms for sharing a physical link among multiple flows, this design adopts SDM with two parallel channels per direction, rather than TDM or virtual-channel-based sharing. SDM physically separates the two channels and therefore avoids virtual-channel allocation logic and per-channel buffering which virtual-channel-based mechanism requires, or an embedded routing table typically required by TDM mechanisms, but doubles the wire count in each direction.
- **Bufferless routers.** A single-cycle bufferless router is adopted instead of a conventional input-buffered router. This choice is motivated by two considerations. First, eliminating input buffers reduces area cost and

avoids the arbitration logic required for virtual-channel management. Second, a bufferless router provides a fixed one-cycle per-hop latency, which simplifies the timing analysis. The cost of this choice is that contention cannot be absorbed inside the network. Instead, potential contention must be controlled at the injection side, as discussed in the section 3.3.

4.1.2 System Architecture Overview

The implemented system consists of 25 tiles arranged as a 5×5 torus, as shown in Fig. 4.1. Each system node contains a tile and a NoC node. The tile produces and consumes packets, while the NoC node is responsible for transporting them through the network. Each NoC node is further divided into a NI, which mediates between the tile and the network, and a router, which forwards flits along the torus topology.

The router has five logical directions: east, west, north, south, and tile. The tile direction is used for local packet injection and ejection. Each of the four cardinal directions contains two independent bidirectional channels operating in parallel. In hardware, each bidirectional channel is implemented as two unidirectional links, one in each direction. Therefore, each cardinal side of a router exposes two incoming links and two outgoing links. The tile side contains one bidirectional channel for injection and ejection.

Packets are transferred flit by flit. As shown in Table 4.1, each flit consists of a six-bit routing header, a 64-bit payload, and a one-bit `last` signal, which indicate whether this flit is the last flit of this packet. The routing header encodes the information required by dimension-ordered shortest-path routing: a one-bit direction field and a two-bit remaining-hop count for each dimension. The tile sends flits into the NI at a rate of one flit per cycle. The NI buffers the flits across the clock-domain boundary and then injects them into the router. Since the NoC itself is bufferless, once a flit enters the router pipeline, it advances every cycle until it reaches its destination under the contention-free injection conditions derived in Chapter .3.

Bit format for one channel					
1 bit	2 bits	1 bit	2 bits	64 bits	1 bit
x_direction	x_distance	y_direction	y_distance	payload	last

Table 4.1: Flit format for one channel.

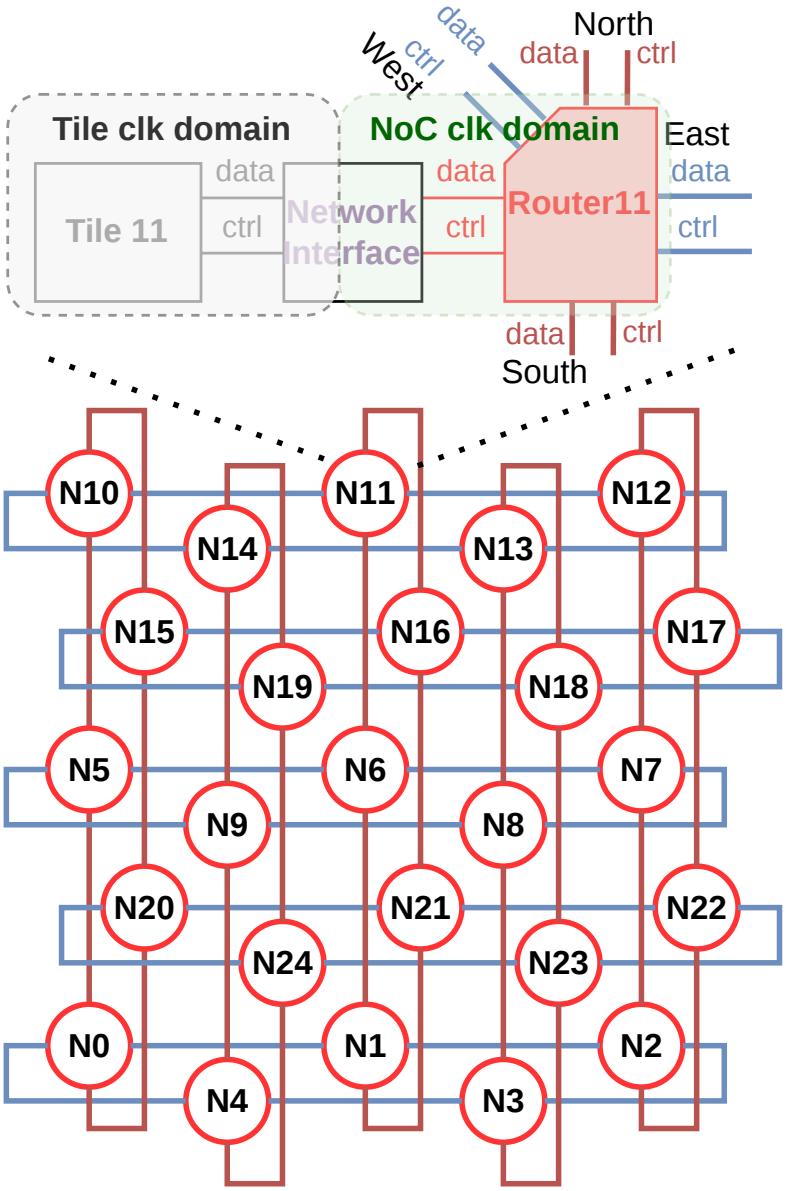


Figure 4.1: System architecture of the 5 × 5 torus NoC.

4.1.3 Router

The router is a single-cycle bufferless design. As illustrated in Fig. 4.2, it consists of four main components: a preprocessing unit, an arbiter, a crossbar, and output registers.

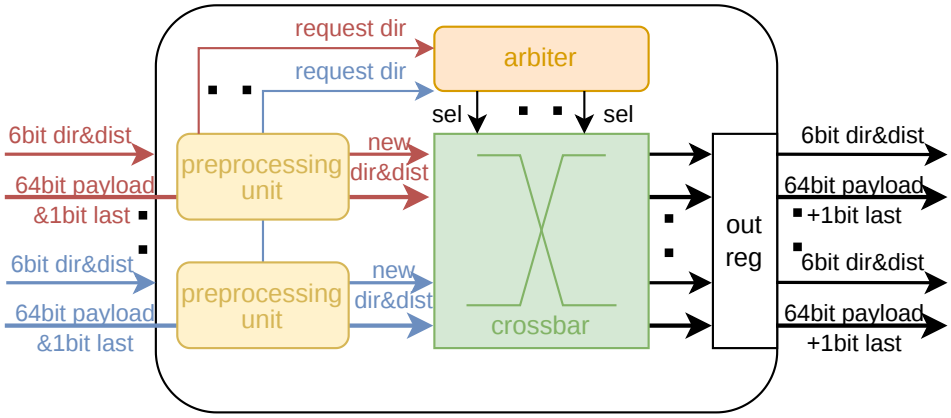


Figure 4.2: Router microarchitecture.

The preprocessing unit determines the output direction of each incoming flit according to its routing header and updates the header for the next hop. Since the routing strategy is dimension-ordered shortest-path routing, the preprocessing logic is implemented as a small combinational block. It checks the remaining hop counts in the X and Y dimensions and decrements the hop count of the dimension currently being traversed, as summarized in Algorithm 1.

The updated header, together with the payload and the `last` signal, is then forwarded to the crossbar. As a result, the next router receives a flit whose remaining-hop count has already been decremented for the dimension traversed in the current hop.

The arbiter, shown in Fig. 4.3, assigns output channels to incoming flits. For each output direction, it collects the requests generated by the preprocessing units of all input ports and selects the corresponding input flits for the available output channels. Since each cardinal direction provides two parallel channels, up to two flits can be forwarded in the same direction in the same cycle. When two requests target the same output direction, the arbiter maps them to the two available channels using a fixed-priority rule. If more requests are presented than the two output channels can serve, only the selected requests are forwarded. In the intended operating regime, such

Algorithm 1 Routing calculation for each input port

Require: flit header $(x_{\text{dir}}, x_{\text{hop}}, y_{\text{dir}}, y_{\text{hop}})$
Ensure: output direction *out* and updated header

if $x_{\text{hop}} > 0$ {the x dimension is not aligned with the destination} **then**
  if $x_{\text{dir}} = 1$ **then**
    $out \leftarrow$ **east**
  else
    $out \leftarrow$ **west**
  end if
  $x_{\text{hop}} \leftarrow x_{\text{hop}} - 1$
else if $y_{\text{hop}} > 0$ {the y dimension is not aligned with the destination} **then**
  if $y_{\text{dir}} = 1$ **then**
    $out \leftarrow$ **north**
  else
    $out \leftarrow$ **south**
  end if
  $y_{\text{hop}} \leftarrow y_{\text{hop}} - 1$
else
  $out \leftarrow$ **tile** {the flit has reached its destination}
end if

oversubscription is avoided by the traffic schedule and the local injection-control strategy. This rule is intentionally simple and deterministic, which keeps the arbitration logic shallow. However, it also introduces a known asymmetry because one channel can be favored over the other. The resulting imbalance in per-channel link utilization is evaluated in the next chapter.

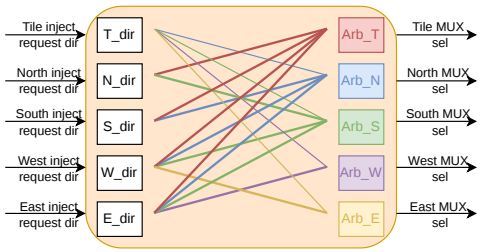


Figure 4.3: Arbiter microarchitecture.

The crossbar implements the connections selected by the arbiter. Because the dimension-ordered routing rule restricts which input directions may legally request each output direction, the crossbar does not need to be implemented as

a fully general N-to-N switch. Instead, the multiplexers are sized according to the valid routing cases.

An X-dimension output port, either east or west, may receive a flit from the tile input or from the opposite X-direction input. Therefore, each X-dimension output channel requires a three-to-one multiplexer, resulting in four such multiplexers across the two X directions and two channels. A Y-dimension output port may receive a flit from the tile input, from the opposite Y-direction input, or from either X-direction input. Therefore, each Y-dimension output channel requires a seven-to-one multiplexer, resulting in four such multiplexers in total. Finally, the tile-ejection port may receive a flit from any of the four cardinal directions and either channel, so each ejection channel requires an eight-to-one multiplexer. Since there are two ejection channels, the router contains two such multiplexers. Overall, the crossbar consists of four three-to-one multiplexers, four seven-to-one multiplexers, and two eight-to-one multiplexers.

4.1.4 Network Interface

The NI bridges the tile and the router across two clock domains. As shown in Fig. 4.4, the injection path consists of two asynchronous First-In, First-Outs (FIFOs)s, an encoder, and a local FSM. The ejection path is implemented using an asynchronous FIFO.

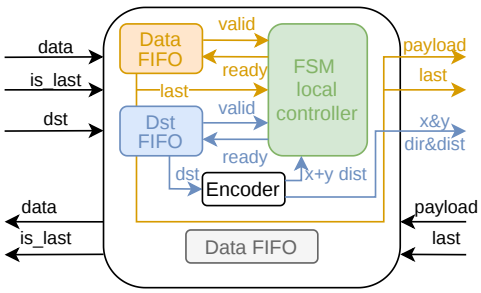


Figure 4.4: Network interface architecture.

Data and destination information are carried on two separate paths. The data FIFO accepts one flit per cycle from the tile when the input valid signal is asserted. It also stores a one-bit `last` signal, which marks the last flit of each packet. The dst FIFO accepts one destination node identifier per packet. Since only one destination entry is required for a packet containing $n_{\text{flit}} \in [8, 192]$ flits, the destination FIFO is much shallower than the data FIFO. In practice, it

mainly serves as a clock-domain crossing buffer rather than as a deep storage structure.

A combinational encoder is connected to the read port of the destination FIFO. Given the destination node identifier, the encoder generates the flit header, including the direction and remaining-hop fields for both dimensions. It also computes the Manhattan distance to the destination, which is used by the local injection controller.

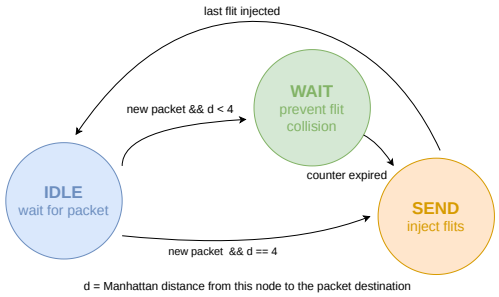


Figure 4.5: Finite state machine for local injection control.

The local injection controller is a three-state FSM, as shown in Fig. 4.5. It controls when packets are injected into the router. In the **idle** state, the controller waits for a valid entry in the destination FIFO, which indicates the start of a new packet. When a new destination becomes available, the controller reads the Manhattan distance from the encoder. If a waiting period is required to avoid packet loss due to contention, the controller transitions to the **wait** state; otherwise, it transitions directly to the **send** state.

The waiting period follows the local worst-case strategy derived in the previous chapter. Specifically, the controller waits for $4 - d$ cycles, where d is the Manhattan distance to the destination. The WAIT state is implemented as a down-counter. When the counter reaches zero, the controller transitions to the SEND state. In the SEND state, the controller forwards one flit per cycle from the data FIFO to the router whenever the FIFO output is valid. The controller returns to the IDLE state in the cycle in which the `last` signal is asserted.

4.2 RTL Verification Testbench

The proposed Network-on-Chip is verified at the RTL level using a self-checking SystemVerilog testbench. The testbench exercises the complete 5×5 NoC under permutation traffic, validates every delivered flit against a

reference model, and reports both correctness and performance metrics in a single simulation run. This section describes the overall testbench structure, the traffic-generation mechanism, the dynamic-frequency controller, and the remaining verification components.

4.2.1 Overall Structure

As shown in Fig. 4.6, the testbench instantiates the Design Under Tests (DUTs) once and surrounds it with several concurrent processes. This lightweight architecture is sufficient because the DUT is driven by a statically generated traffic schedule rather than by unconstrained random transactions at runtime.

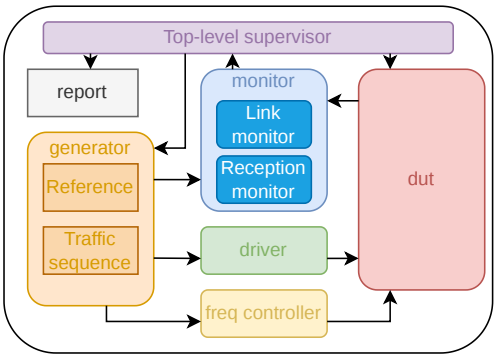


Figure 4.6: Architecture of the RTL verification testbench.

The main testbench processes are as follows:

1. **Generator**, which constructs the traffic schedule and derives the reference model used for output checking.
2. **Driver**, which drives the per-tile injection interfaces in the tile clock domain.
3. **Monitor**, which observes the ejection interfaces for functional checking and samples internal link-valid signals for utilization measurement.
4. **Dynamic-frequency controller**, which adjusts the NoC clock period according to the current packet-length condition when dynamic frequency control is enabled.
5. **Top-level supervisor**, which coordinates reset, simulation completion, timeout checking, and final reporting.

The DUT contains two clock domains. The `tile_clk` domain models the tile-side interface that supplies flits to each tile's injection FIFO. The `noc_clk` domain clocks the routers and the NoC-side ejection logic. The testbench generates and drives these two clocks independently, which allows both static-frequency and dynamic-frequency experiments to be evaluated.

4.2.2 Traffic Models and the Generator

The generated traffic is constrained by three rules:

1. The source is not allowed to send a packet to itself.
2. The same source cannot choose the same destination in two consecutive packets.
3. The schedule should prevent two simultaneously active packets from targeting the same destination ejection port.

Two traffic models are used throughout the evaluation, following the permutation-style communication assumption that motivates the proposed design.

The first model is the *deterministic* model. It uses a deterministic sequence of four cyclically rotating shift permutations on the 5×5 torus:

$$\{(+2,+2),(+2,+3),(+3,+2),(+3,+3)\} \pmod 5.$$

All packets in this model travel the maximum shortest-path distance of four hops and use the minimum packet length of eight flits. This model therefore creates the tightest operating point for the design: it combines the longest routing path and the highest per-packet overhead ratio. It is used both as a functional stress test and as a sanity check for the analytically derived NoC frequency requirement.

The second model is the *random* model. It generates a sequence of independent permutation-style communication patterns, where the destination is random under the constraints. Packet length can either be fixed at a specified number of flits or randomly selected from the range $[8, 192]$ flits. This model represents more typical but still tight operating conditions and is used to evaluate link utilization under both static and dynamic frequency schemes.

A central design choice of the testbench is that the complete traffic schedule is generated *a priori*, before the main simulation begins. This decouples traffic generation from DUT execution and provides a deterministic reference for self-checking. In the random traffic mode, the generator constructs the schedule

through a sequence of allocation iterations. In each iteration, it identifies the sources whose previously assigned packets have completed and assigns each of them a new destination from the currently available receivers, subject to the constraints described above. The generation procedure is summarized in Fig. 4.7.

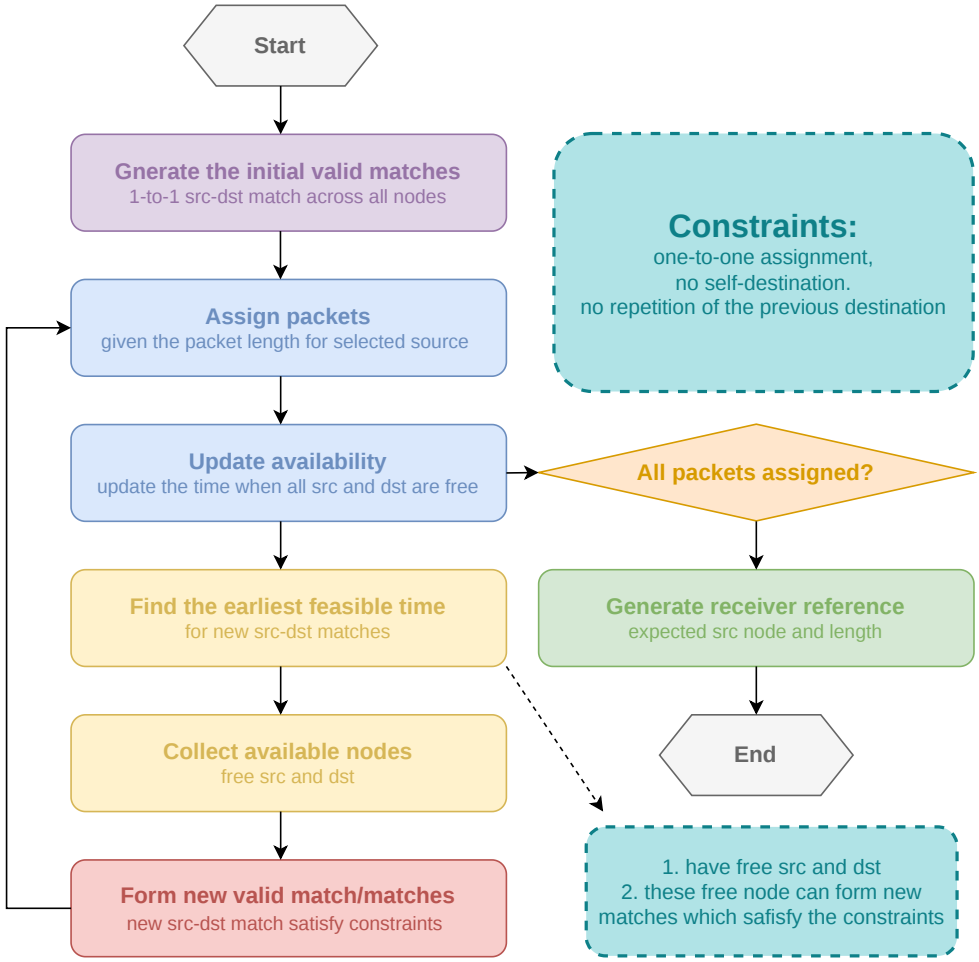


Figure 4.7: Random permutation traffic generation procedure.

Fig. 4.8 shows a snapshot of a generated random traffic pattern. The left plot gives the Transmits (TXs)-side view, where each row corresponds to a source node and the color indicates the destination node of the packet currently being transmitted. The right plot gives the Receives (RXs)-side view, where each row corresponds to a destination node and the color indicates the source node of the packet currently being received.

The snapshot illustrates the properties enforced by the generator. If a vertical slice is taken at any cycle in the TX-side plot, the non-idle colors do not repeat. This means that no two active sources are assigned to the same destination at the same time, which reflects the one-to-one permutation constraint. The RX-side plot shows the same property from the opposite perspective: each active destination receives from at most one source at a given cycle. The colored regions are mostly continuous along the time axis, indicating the traffic is tight. The occasional white regions are expected. They represent idle cycles, which may occur when a source has finished its previous packet but no currently available destination satisfies all scheduling constraints. Therefore, these idle gaps are a normal consequence of constrained random permutation generation rather than an indication of deadlock or incorrect behavior.

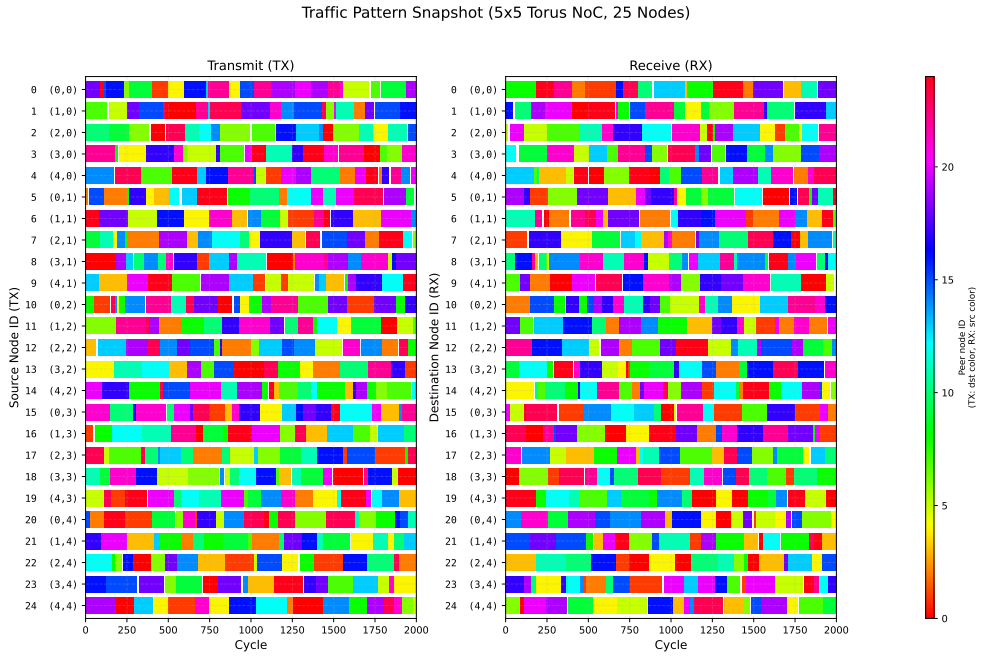


Figure 4.8: Snapshot of generated random permutation traffic.

The generated schedule is stored in `pkt_sched[s][p]`, where each entry records the destination, packet length, and injection time of packet p from source s. From this schedule, the testbench derives a second data structure, `rx_expect_list[d]`, which lists, for each node, the packets expected to be received by that node in order. This structure serves as the reference model for the monitor.

Because generating the random schedule and constructing the reference model can take a non-trivial amount of time for large simulations, the testbench caches both `pkt_sched` and `rx_expect_list` in files. Each cached file is associated with an explicit configuration fingerprint:

$$\texttt{\#CFG}\ \langle X,\ Y,\ P,\ n_{\min},\ n_{\max},\ \textsc{nflits_mode},\ \textsc{traffic_mode},\ n_{\text{wc}}\rangle.$$

At startup, the testbench compares the fingerprint in the cached files with the current simulation parameters. If the fingerprint matches, the cached schedule and reference model are loaded directly. Otherwise, a new schedule is generated and written back to the cached file. This mechanism makes repeated simulations with the same configuration much faster while preserving reproducibility.

4.2.3 Dynamic Frequency Controller

To model the proposed dynamic frequency-scaling scheme, the testbench includes a controller process and a programmable `noc_clk` generator. The controller observes the current in-flight packet of each tile and computes the minimum packet length among all sources that have not completed transmission. Whenever this minimum changes, the target NoC clock period is recomputed as

$$T_{\text{NoC}} \;=\; T_{\text{Tile}} \cdot \frac{n_{\text{flit}}^{\min}}{n_{\text{flit}}^{\min} + N_{\text{ovh}}}, \tag{4.1}$$

where $N_{\text{ovh}} = 4$ accounts for the injection and routing overhead considered in the frequency-scaling model. The `noc_clk` generator then updates its clock period accordingly.

Two frequency-scaling modes are supported. The first mode is a fine-grained mode, in which the NoC frequency is adjusted for each possible value of $n_{\text{flit}}^{\min}$. Since packet lengths range from 8 to 192 flits, this mode can produce up to 185 distinct frequency levels. It is useful for analysis but is less practical for hardware implementation. The second mode is a quantized mode, in which the packet-length range is divided into five logarithmic bins. Each bin maps to one frequency level. This mode is less precise but better represents a practical runtime frequency controller with a small number of supported operating points.

4.2.4 Other Components

Driver. The driver runs in the `tile_clk` domain. On every negative edge of `tile_clk`, it checks the next scheduled packet entry for each of the 25 tiles. If the scheduled injection time has not yet been reached, the corresponding injection interface remains idle. Otherwise, the driver sends one flit into the tile's injection FIFO. Packet boundaries are marked using the `last` signal. The flit payload encodes the tuple

$$\langle \text{src_id}, \text{dst_id}, \text{sequence} \rangle,$$

which allows the monitor to check each received flit field by field. Once a tile has injected the final flit of its final packet, its injection interface is permanently deasserted.

Monitor. The monitor has two responsibilities: reception checking and link-utilization measurement. For reception checking, it runs in the `tile_clk` domain and observes the ejection `valid` signal and the ejected flit at each tile. For every accepted flit, the monitor checks three conditions against `rx_expect_list`: the destination field must match the receiving tile, the source field must match the expected source of the currently active packet at that destination, and the sequence number must match the number of flits already received for that packet.

If any of these checks fails, an error counter is incremented and the failure is logged with the relevant source, destination, sequence, and time information. When the expected number of flits for a packet has been received, the monitor advances the expectation pointer for that destination and increments the global completed-packet count. In this way, the monitor verifies flit integrity, ordering within each packet, and ordering between packets received at the same destination. Only $O(1)$ state is required per destination because the expected order has already been materialized in `rx_expect_list`.

For link-utilization measurement, the monitor samples the internal valid signals of the router links on every `noc_clk` edge through hierarchical references into the DUT. The sampled links include the east, west, north, and south directions, with both channels in each direction. The measurement window is configurable using a start time and an end time, allowing the warm-up and drain phases to be excluded. At the end of the simulation, the monitor reports utilization at multiple granularities, including per-node, per-direction, per-channel, and global-average utilization. Since the monitor only observes DUT signals and does not drive any of them, it does not affect the verified

behavior of the design.

Top-level supervisor. The top-level supervisor controls the overall simulation flow. It applies reset, starts traffic generation or cache loading, launches the concurrent testbench processes, checks for completion, and produces the final report. The simulation can terminate in two ways. The first is natural completion, which occurs when every tile has injected its final packet and every destination has received all packets listed in its `rx_expect_list`. The second is timeout expiration, controlled by `SIM_TIMEOUT`. The timeout mechanism prevents a deadlocked or unexpectedly slow simulation from running indefinitely during development and regression testing.

At the end of the simulation, the supervisor emits a `TEST PASSED` or `TEST FAILED` verdict. The test passes only if no flit-level errors were detected, the total number of injected flits equals the total number of ejected flits, and the total number of completed packets equals the total number of packets in the reference model. If any condition fails, the corresponding mismatch is reported separately so that the failure mode can be identified directly. The final report also includes the link-utilization summary and per-node injection and reception statistics.

4.3 Summary

This chapter presented the RTL implementation and verification framework of the proposed 5×5 torus NoC. On the hardware side, the design adopts two parallel SDM channels per direction and a single-cycle bufferless router to reduce buffering, arbitration complexity, and per-hop latency. The system architecture, router datapath, NI, asynchronous FIFOs, routing-header format, and local injection controller were described in detail. On the verification side, a self-checking SystemVerilog testbench was developed to validate the complete NoC under both fixed and random permutation traffic. The testbench generates reproducible traffic schedules, constructs a reference model, checks received flits for correctness, monitors link utilization, and supports dynamic NoC frequency scaling. Together, the hardware implementation and RTL verification infrastructure provide the basis for the functional and performance evaluation presented in the next chapter.

Chapter 5

Simulation and Hardware Result

This chapter reports the simulation and hardware results of the proposed NoC The simulation is conducted in QuestaSim using the testbench from Chapter 4. Logic synthesis is performed with Synopsys Design Compiler using a 28-nm standard cell library, targeting 2.5 GHz. A preliminary place-and-route experiment is run on the same library, with each tile represented by placement and routing blockages.

Section 5.1 presents the RTL testbench results, covering functional verification and link utilization under static and dynamic frequency control. Section 5.2.1 reports the post-synthesis per-node area and its scaling with the data path width, and Section 5.2.2 reports the post-layout operating point obtained from the preliminary physical design.

5.1 RTL Testbench Results

This section reports the RTL simulation results obtained from the testbench described in Section 4.2. Each experiment injects 50 000 packets per node, corresponding to 1.25×10^6 packets over the full 5×5 network. The following configurations are evaluated:

- **Static frequency, deterministic permutation traffic.** The deterministic permutation traffic mode of Section 4.2.2 is used. All packets travel $d = 4$ hops and contain $n_{\text{flit}} = 8$ flits. This case is used for functional correctness verification.
- **Static frequency, random permutation traffic, fixed length.** Random

permutation traffic is generated with a fixed packet length of $n_{\text{flit}} = 8$. The NoC runs at the static worst-case frequency of 1.5× the tile frequency.

- **Static frequency, random permutation traffic, random length.** Random permutation traffic is generated with packet lengths uniformly distributed in $[8, 192]$. The NoC still runs at the static 1.5× frequency.
- **Fine-grained dynamic frequency, random permutation traffic, random length.** The same mixed-length random traffic is used, but the NoC frequency is adjusted according to the active packet-length information. The ideal period scaling follows $n_{\text{flit}}/(n_{\text{flit}} + 4)$, or equivalently the normalized frequency follows $(n_{\text{flit}} + 4)/n_{\text{flit}}$.
- **Quantized dynamic frequency, random permutation traffic, random length.** The dynamic frequency is quantized to five discrete frequency levels. The levels are selected according to logarithmically spaced packet-length ranges indexed by $\lfloor \log_2 n_{\text{flit}} \rfloor$, which represents a more hardware-realistic controller implementation.

The reported link utilization excludes the warm-up and drain phases of the simulation. It is averaged over all cardinal-direction physical links of the 5×5 torus. Since each physical direction contains two parallel channels, the testbench directly records the active-cycle fraction of channel 0 and channel 1, denoted as u_{ch0} and u_{ch1}, respectively. Due to the static-priority channel allocation strategy, channel 1 is only used when channel 0 is already occupied for the same physical direction. Therefore, the fraction of cycles in which both channels are active is

$$u_{\text{dual}} = u_{\text{ch1}}, \tag{5.1}$$

and the fraction of cycles in which exactly one channel is active is

$$u_{\text{single}} = u_{\text{ch0}} - u_{\text{ch1}}. \tag{5.2}$$

The two-channel link utilization is computed as the average occupancy of the two channels:

$$u_{\text{avg}} = \frac{u_{\text{ch0}} + u_{\text{ch1}}}{2} = \frac{u_{\text{single}} + 2u_{\text{dual}}}{2}. \tag{5.3}$$

5.1.1 Functional Verification

Functional correctness is first checked using the reception monitor in the RTL testbench. The monitor compares every received flit against the expected

source node, destination node, packet sequence, and intra-packet flit order. Across the verification runs, the monitor reports zero flit-level errors: every observed flit reaches the correct destination, the ordering within each packet is preserved, and the total number of ejected flits matches the total number of injected flits.

Fig. 5.1 shows a representative waveform for one routed packet. In this example, the first flit of a packet is injected from node 0 and delivered to node 18. Under the torus routing rule, it is first forwarded from node 0 to node 4 through the west output port, and then continues west to node 3, moves north to node 23 through the wrap-around connection, and finally moves north again to node 18, where it is ejected through the local tile port. This waveform confirms that the router logic, inter-router link connection and wrap-around torus behavio are consistent with the expected routing path.

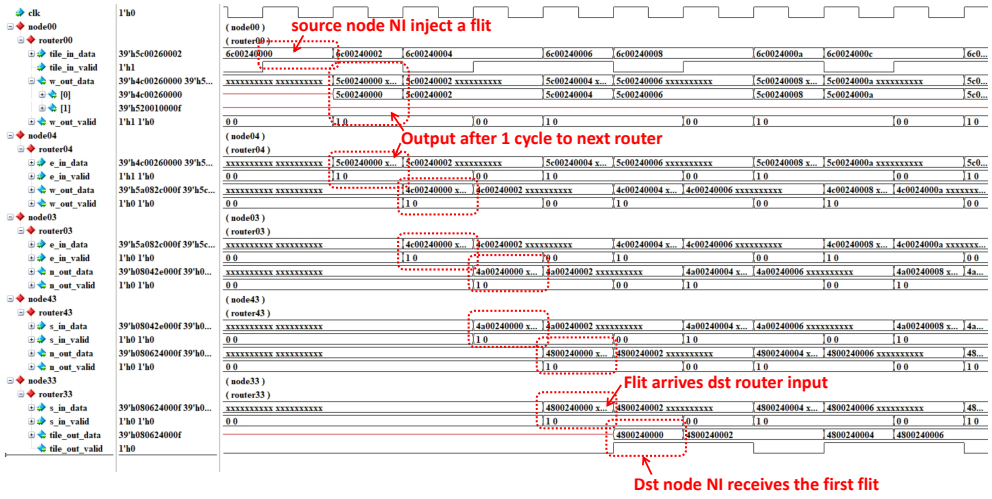


Figure 5.1: QuestaSim waveform for a routed packet from node 0 to node 18.

5.1.2 Static-Frequency Link Utilization

The fixed-length random traffic case provides the baseline for evaluating the static-frequency design. With eight-flit packets and a static NoC frequency of 1.5× the tile frequency, the measured average link utilization is 20.8%. The measured channel utilization is 38.1% for channel 0 and 3.5% for channel 1. Therefore, 3.5% of link-cycles use both channels simultaneously, while $38.1\% - 3.5\% = 34.6\%$ of link-cycles use exactly one channel.

This measured utilization is consistent with a simple utilization estimate based on the traffic pattern, packet overhead, and average path length. Three factors determine the estimate. First, under random permutation traffic on the 5×5 torus, the 25 source–destination flows are distributed over the available link resources. The network contains 50 inter-node connections, each comprising two bidirectional channels, and each channel provides two unidirectional links, yielding $50 \times 2 \times 2 = 200$ unidirectional links in total. In the worst case, every flow traverses a maximal path of $d_{\max} = 4$ hops, occupying $25 \times 4 = 100$ links. The maximum average link utilization is therefore bounded by $100/200 = 50\%$ of the available link capacity, which gives the upper bound of the link utilization. Second, the static NoC frequency is provisioned for the worst-case eight-flit packet. Since the routing and synchronization overhead contributes four additional cycles, only $8/(8+4) = 8/12$ of the elapsed NoC cycles carry useful packet flits. Third, under uniform random destinations on the 5×5 torus, the average shortest-path distance is $\bar{d} = 2.5$ hops, while the worst-case shortest-path distance is $d_{\max} = 4$ hops. The average link occupancy therefore scales by $\bar{d}/d_{\max} = 2.5/4$. Combining these factors gives

$$u_{\text{theory}} = \underbrace{0.5}_{\text{topology}} \times \underbrace{\frac{8}{12}}_{\text{flit/overhead ratio}} \times \underbrace{\frac{2.5}{4}}_{\text{average/worst-case distance}} = 0.2083 = 20.83\%. \tag{5.4}$$

The analytical estimate matches the measured 20.8% result to within rounding error, which is consistent with both the traffic generator behavior and the utilization accounting method.

The large difference between the two channel counters is also expected. Channel 0 is selected first by the static-priority arbitration strategy described in Section 4.1.3, so most routed flits use channel 0 whenever it is available. Channel 1 is only activated when channel 0 is already occupied by another flit requiring the same output direction. Therefore, channel 1 has a much lower utilization than channel 0, even though the aggregate two-channel capacity is available.

When the packet length is randomized uniformly in $[8, 192]$ while the NoC frequency remains fixed at 1.5×, the average utilization decreases slightly to 20.4%. The measured channel utilization is 37.9% for channel 0 and 3.0% for channel 1, giving 34.9% single-channel link-cycles and 3.0% dual-channel link-cycles. This reduction is caused by scheduling gaps introduced by the mixed-length traffic. In the fixed-length case, all sources tend to complete packets at the same time, so new permutation rounds can start in a more

synchronized manner. In the mixed-length case, short packets finish much earlier than long packets. A source that has completed its packet may need to wait before its next destination becomes available, because other sources may still be transmitting longer packets to destinations in the current permutation set. These occasional injection-side gaps reduce the number of useful link cycles and explain the small drop relative to the fixed-length baseline.

5.1.3 Dynamic-Frequency Link Utilization

The dynamic-frequency experiments use the same mixed-length random traffic as the static mixed-length case. The purpose is to evaluate whether the NoC can improve effective link utilization by avoiding the fixed worst-case frequency setting. Two dynamic policies are considered. The fine-grained strategy selects the NoC frequency using the current limiting active packet length, denoted as $n_{\min}$. The corresponding normalized frequency is

$$f_{\text{norm}} = \frac{n_{\min} + 4}{n_{\min}}. \tag{5.5}$$

A smaller $n_{\min}$ therefore requires a higher NoC frequency, while a larger $n_{\min}$ allows the NoC frequency to move closer to the tile frequency. The quantized strategy approximates this continuous behavior using only five discrete frequency levels, which is closer to what a practical clock divider or Phase-Locked Loops (PLLs)-based controller could support.

Under the fine-grained dynamic strategy, the average link utilization increases to 26.5%. The channel-level utilization is 47.2% on channel 0 and 5.9% on channel 1. Therefore, 5.9% of link-cycles use both channels simultaneously, while $47.2\% - 5.9\% = 41.3\%$ of link-cycles use exactly one channel. Compared with the static mixed-length baseline of 20.4%, the fine-grained scheme improves average link utilization by 6.1 percentage points, corresponding to an approximately 30% relative improvement.

Fig. 5.2 shows how the fine-grained controller distributes simulation time across different limiting packet lengths and frequency states. The upper plot reports the dwell percentage for each observed $n_{\min}$ value, where $n_{\min}$ is the minimum packet length among the currently active transfers and therefore determines the global NoC frequency. The distribution first increases and then decreases, which is consistent with the behavior of the dynamic strategy. For very small $n_{\min}$ values, the selected frequency is high, but these short packets finish quickly and therefore do not occupy a large fraction of the total simulation time. For very large $n_{\min}$ values, although the corresponding

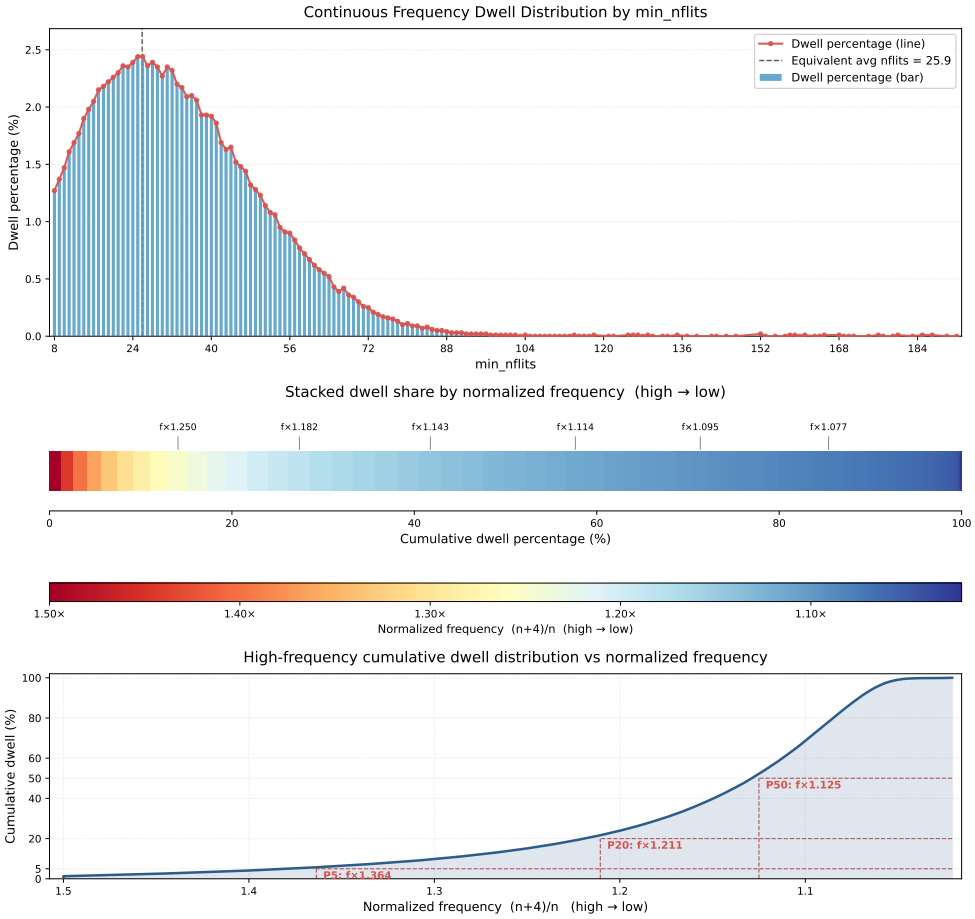


Figure 5.2: Dwell-time distribution for different frequencies and the corresponding packet lengths under fine-grained dynamic frequency scaling.

packets are long, they rarely dominate the global frequency because long transfers are likely to coexist with shorter active transfers. Once a shorter transfer exists, the global frequency is determined by that shorter packet instead of the long one. As a result, the dwell percentage becomes very small in the large-$n_{\min}$ region, and most of the dwell time is concentrated in the small-to-medium $n_{\min}$ range.

The middle plot presents the same dwell information in the normalized-frequency domain, accumulated from high frequency to low frequency. Since the normalized frequency is $(n_{\min}+4)/n_{\min}$, smaller $n_{\min}$ values correspond to higher-frequency states. The color distribution shows that the high-frequency region only occupies a small portion of the total dwell time, while most of the simulation time is accumulated after the frequency has already dropped well below the static worst-case value of 1.5×.

The lower plot quantifies this trend using the high-frequency cumulative dwell distribution. Only about 5% of the dwell time is spent at frequencies above 1.364×, and only about 20% is spent above 1.211×. The median point is around 1.125×, meaning that half of the total dwell time occurs at frequencies no higher than approximately 1.125×. Therefore, although the dynamic scheme still raises the NoC frequency when short active packets require it, these high-frequency intervals are limited. This is the main reason why fine-grained dynamic scaling improves the effective link utilization compared with the static worst-case frequency setting.

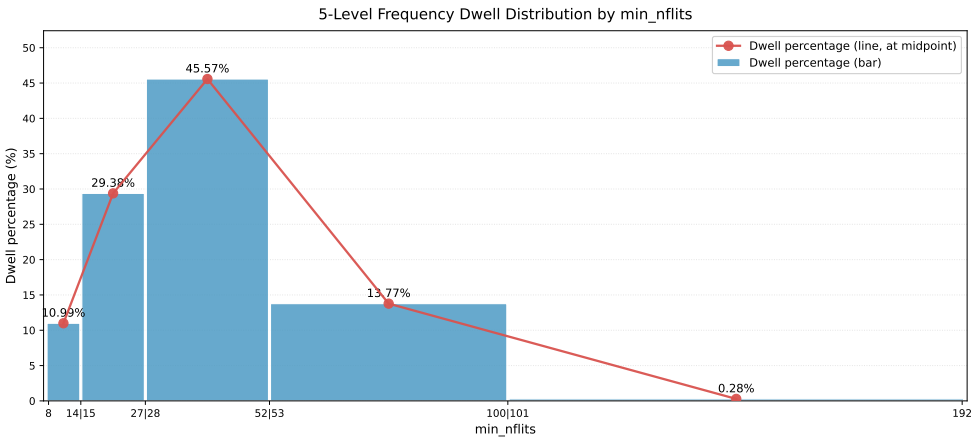


Figure 5.3: Dwell-time distribution for different packet length levels under five-level quantized dynamic frequency scaling.

The five-level quantized case achieves 25.4% average link utilization, with

45.4% utilization on channel 0 and 5.3% utilization on channel 1. Therefore, 5.3% of link-cycles use both channels simultaneously, while $45.4\% - 5.3\% = 40.1\%$ of link-cycles use exactly one channel. This result is 1.1 percentage points below the fine-grained case, but still 5.0 percentage points above the static mixed-length baseline. The gap between the fine-grained and quantized policies is the cost of discretization: when the ideal frequency falls between two available levels, the controller must select a conservative discrete operating point, so some cycles remain over-provisioned compared with the continuous reference. The corresponding dwell-time distribution is shown in Fig. 5.3.

Table 5.1: Measured link utilization across the random-traffic configurations.

Configuration	Average (%)	Channel 0 (%)	Channel 1 (%)
static, fixed length	20.8	38.1	3.5
static, mixed length	20.4	37.9	3.0
dynamic, fine-grained	26.5	47.2	5.9
dynamic, 5-level	25.4	45.4	5.3

5.2 Hardware Result

5.2.1 Logic Synthesis

The design was synthesized with Synopsys Design Compiler targeting a clock frequency of 2.5 GHz on the 28-nm standard-cell library. Four payload widths were swept, $W \in \{16, 32, 64, 128\}$ bits, to characterize how the area cost of each architectural component scales with the data path. All four runs close timing at the 2.5 GHz target. The post-layout frequency is substantially lower than this synthesis-stage figure and is discussed separately in Section 5.2.2; the synthesis results reported here therefore represent an optimistic upper bound that isolates the logical cost of the design from interconnect parasitics.

Per-Node Area and Composition

The per-node area grows from approximately $2\,000\,\mu\text{m}^2$ at $W = 16$ to $12\,000\,\mu\text{m}^2$ at $W = 128$, corresponding to a total network area of 0.054 mm^2 and 0.31 mm^2, respectively, for the full 5×5 design. Across all four widths, the NI accounts for 50 to 55% of the per-node area, and the router for the remaining 45 to 50%. The more detailed results are shown in Fig .5.4 and

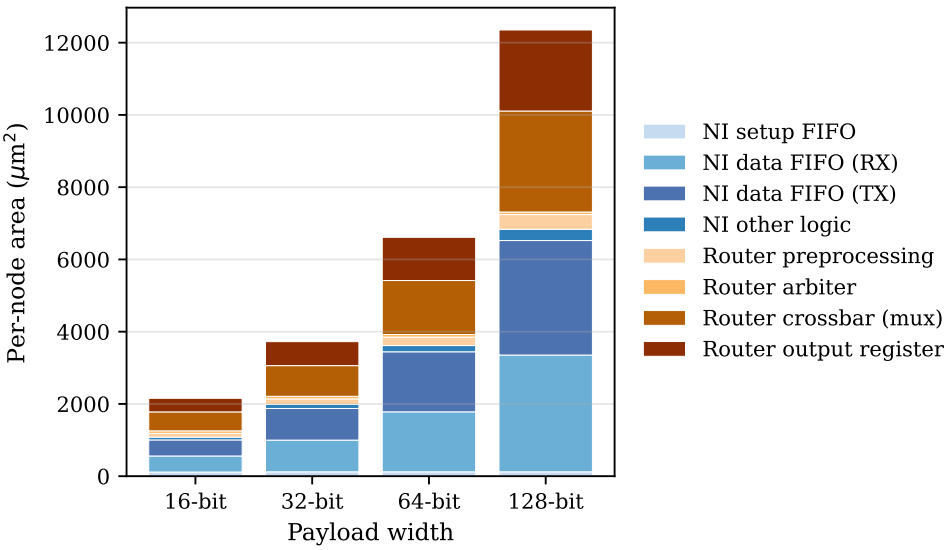


Figure 5.4: Component-level area breakdown of a NoC node under different payload widths.

Fig .5.5, which demonstrate that the crossbar and the output registers take the main area of the router, while two asynchronous data FIFOs, one for TX and one for RX , take the main area of the NI. Besides, as we can expect, these components consistently increasing as data path width increasing, .

Width Scaling

The payload-dependent part of the design is expected to increase approximately linearly with the data path width, while the control-dominated logic should remain nearly unchanged. To quantify this trend, a linear model

$$A = aW + b \tag{5.6}$$

is fitted for each component across the four data path widths $W \in \{16, 32, 64, 128\}$, where A is the synthesized area, a is the area increase per payload bit, and b is the width-independent area component. The fitted parameters are reported in Table 5.2, and the corresponding scaling curves are shown in Fig. 5.6.

The fitted results clearly separate the design into control-dominated components and payload-width-dependent components. The Dst FIFO and the arbiter have almost zero slope, with $a = 0.035\,\mu\text{m}^2/\text{bit}$ and $a =$

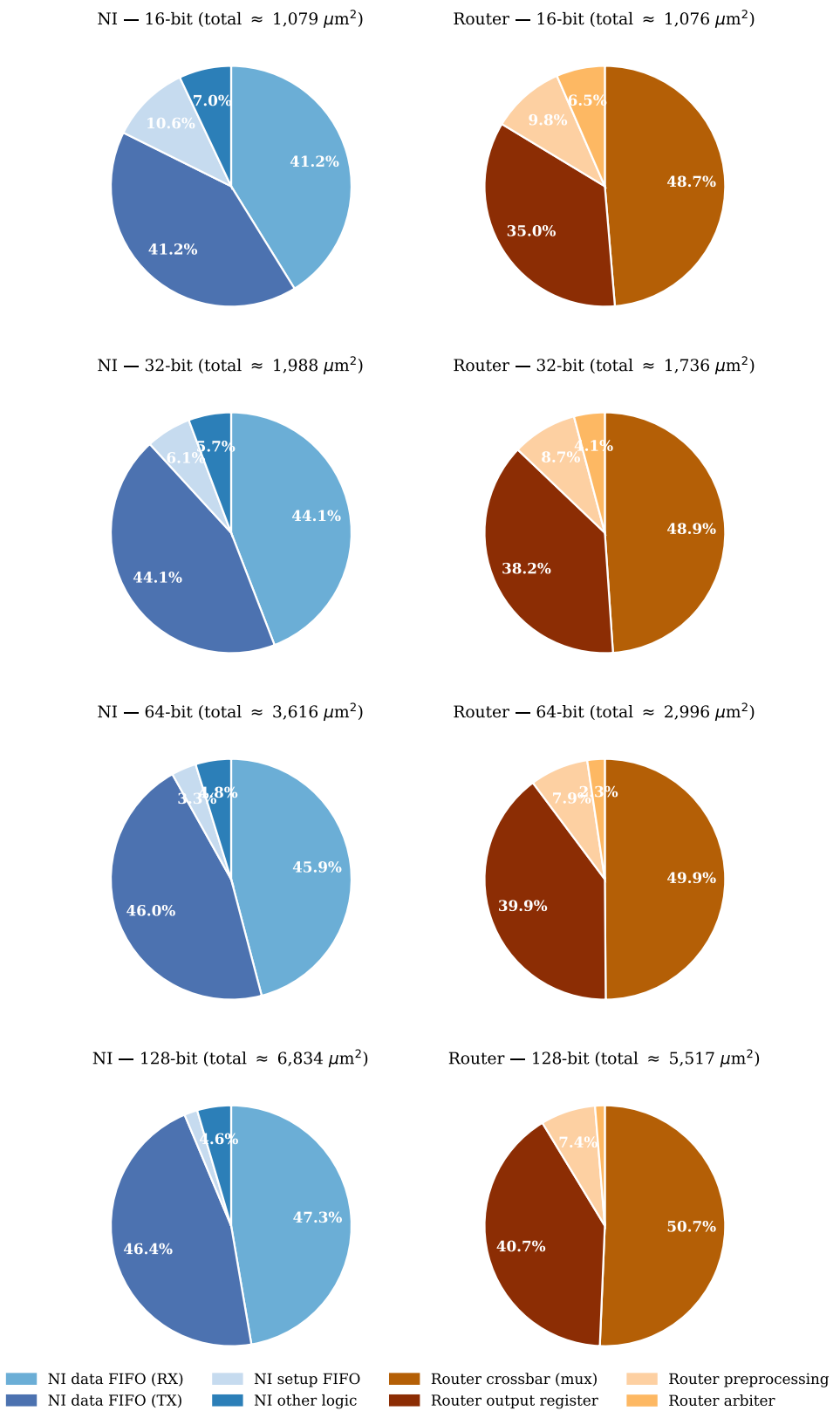


Figure 5.5: Internal area proportion breakdown of the network interface and router under different payload widths.

Table 5.2: Per-component linear fit $A = aW + b$ across $W \in \{16, 32, 64, 128\}$.

Component	Slope a (μm^2/bit)	Intercept b (μm^2)	R^2
NI Dst FIFO	0.035	117.283	0.329
NI Data FIFO (RX)	24.777	67.005	1.000
NI Data FIFO (TX)	24.193	86.210	1.000
NI other logic	2.086	43.279	0.999
Router Preprocessing unit	2.679	64.219	1.000
Router Arbiter	0.011	70.390	0.306
Router Crossbar	20.277	199.035	1.000
Router Output registers	16.597	123.737	1.000
Router other logic	-0.001	0.240	0.254
Node (total)	90.653	771.870	1.000
NI (total)	51.091	313.777	1.000
Router (total)	39.564	457.621	1.000

0.011 μm^2/bit, respectively. Their low R^2 values do not indicate strong nonlinear scaling; instead, they mainly reflect that their area is nearly constant across the tested widths, so small synthesis variations dominate the fitted trend. This behavior is consistent with their functions. The Dst FIFO stores fixed-width destination and routing metadata, while the arbiter operates on request and grant signals rather than payload data. Their area is therefore mainly represented by the intercept term, around 117 μm^2 for the Dst FIFO and 70 μm^2 for the arbiter.

The NI area is strongly width-dependent because it contains the two payload data FIFOs. The Data FIFO (RX) has a slope of 24.777 μm^2/bit, and the Data FIFO (TX) has a similar slope of 24.193 μm^2/bit. Together, these two FIFOs contribute approximately 48.97 μm^2/bit, which accounts for most of the NI (total) slope of 51.091 μm^2/bit. The remaining NI overhead has a much smaller slope of 2.086 μm^2/bit, indicating that only a limited amount of width-dependent logic exists outside the main data storage structures. As a result, the fitted NI area increases from 1131.228 μm^2 at 16 bits to 6853.388 μm^2 at 128 bits.

The router area is also close to linear with the data path width. Its width-dependent area is mainly caused by the crossbar data path. The Crossbar contributes 20.277 μm^2/bit, and the Output registers contribute another 16.597 μm^2/bit. These two blocks together contribute 36.874 μm^2/bit, which is the dominant part of the Router (total) slope of 39.564 μm^2/bit. The Preprocessing unit also has a smaller but still linear slope of 2.679 μm^2/bit.

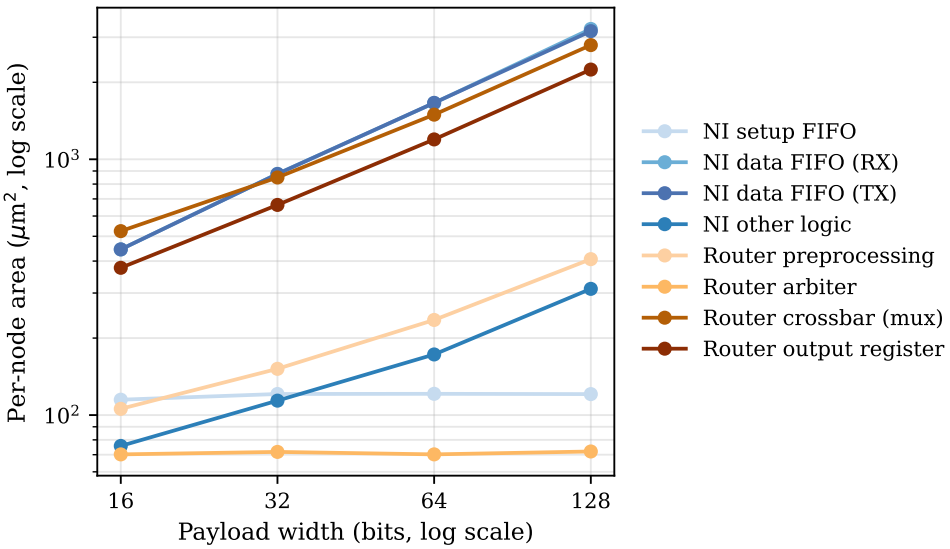


Figure 5.6: Area scaling with data path width for the synthesized NoC components.

Therefore, the router area increases from 1090.639 μm^2 at 16 bits to 5521.768 μm^2 at 128 bits.

At the full node level, the fitted slope is 90.653 μm^2/bit, which is almost exactly the sum of the NI (total) slope and Router (total) slope. This confirms that the node-level width scaling is dominated by the payload storage and switching structures: the two data FIFOs in the NI, the Crossbar, and the Output registers. Across the tested range, the full node area increases from 2222.312 μm^2 at 16 bits to 12375.404 μm^2 at 128 bits, and the near-unity R^2 shows that a linear model accurately describes the area scaling behavior of the complete node.

5.2.2 Physical Design

To evaluate the feasibility of the proposed NoC beyond synthesis-level area and timing estimates, a preliminary placement-and-routing experiment is performed on the full 5 × 5 network. This physical design is not intended to represent a sign-off-grade implementation, but it provides a more realistic post-layout operating point by including floorplanning constraints, placement, routing, and interconnect parasitics.

Fig. 5.7 shows the floorplan used in this experiment. Each system node

consists of one tile region and one corresponding NoC node. Since no real tile macro is available in this work, each tile is represented by a fixed-size placeholder occupying a 1 000 μm × 1 000 μm region. Placement blockages are inserted over these tile regions to prevent standard cells from being placed inside them, and routing blockages are also applied so that signal wires are forced to route around the tile placeholders. Each NoC node, including both the router and the NI, is constrained to a fixed 150 μm×150 μm region adjacent to its associated tile. A 250 μm spacing is inserted between neighboring system nodes to provide routing channels and placement margin. The resulting die size is 7 mm × 7 mm, which is sufficient to accommodate all 25 tile placeholders, NoC nodes, and inter-node routing resources.

A folded torus layout is adopted to reduce the physical length of the logical wrap-around links compared with a direct torus placement. In a direct placement, the wrap-around links would connect nodes located near opposite sides of the array, resulting in long interconnects. The folded arrangement instead maps these logical neighbor connections to regular inter-node links with a bounded physical span, typically on the order of two tile pitches. However, due to the large tile size, these folded inter-node links can still incur significant wire delay. Therefore, the post-layout timing remains strongly affected by NoC link routing, even though the longest wrap-around distances are reduced.

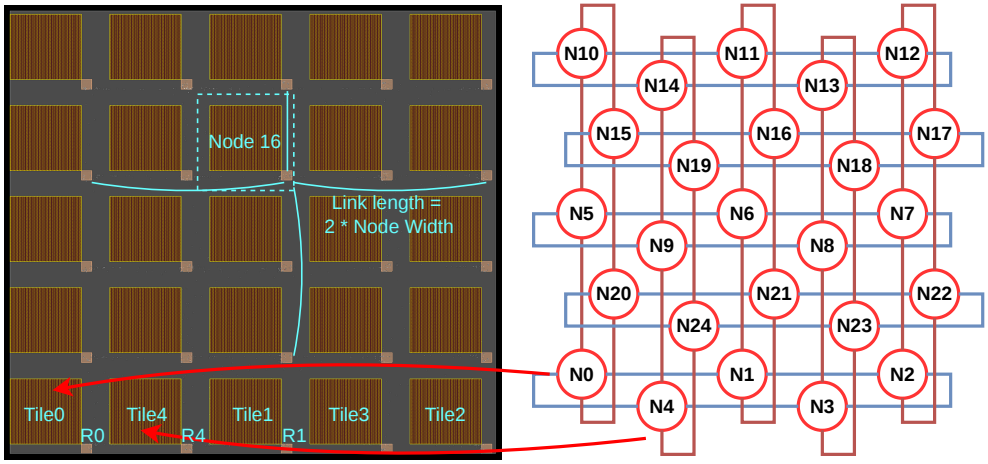


Figure 5.7: Floorplan of the 5 × 5 NoC with tile placeholders and adjacent NoC node regions.

After placement and routing, the post-layout snapshot is shown in Fig. 5.8. The NoC clock domain closes timing at approximately the 800 MHz level.

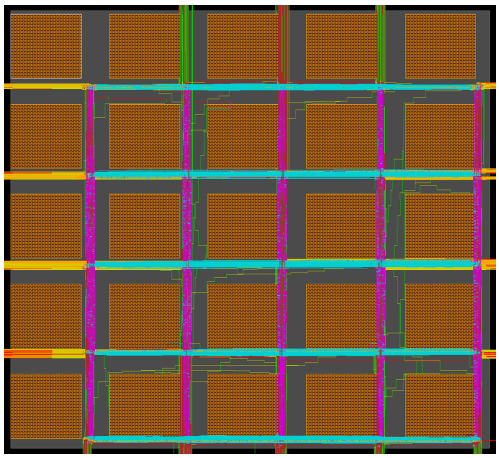


Figure 5.8: Post-layout placement-and-routing snapshot of the full 5×5 NoC.

This frequency is noticeably lower than the pre-layout timing result obtained from Design Compiler. The difference is expected for two main reasons. First, the synthesis flow was performed without a wire-load model, so the pre-layout timing estimate did not include interconnect parasitics and was therefore optimistic. Second, in the physical floorplan, the folded inter-node links have a bounded but still substantial physical span. In this implementation, the longest inter-node connection is on the order of $2 \times 1,400, \mu$m before routing detours around tile placeholders and blockages are considered, which introduces significant wire delay on the critical paths. The relatively large spacing between system nodes further increases the routing length and contributes to the post-layout frequency reduction.

Although the post-layout frequency is lower than the synthesis-only estimate, the result still supports the system-level frequency assumption used earlier in this work. With the 1.5× NoC-to-tile frequency ratio derived in Section 3.4, a NoC frequency near 800 MHz corresponds to a tile frequency of approximately 500 MHz. This operating range remains practical for typical synchronous digital tile logic. Therefore, while the preliminary physical design exposes the importance of interconnect delay and floorplan constraints, it does not invalidate the feasibility of the proposed NoC architecture under the simplified physical-design assumptions used in this thesis.

5.3 Summary

This chapter presented the simulation, synthesis, and preliminary physical-design results of the proposed 5×5 torus NoC. Functional verification confirmed correct end-to-end delivery across 1.25×10^6 packets, with zero flit-level errors in source, destination, sequence, and intra-packet ordering. The measured link utilization matched the analytical model to within rounding error under static frequency, and the dynamic frequency schemes raised the average utilization from 20.4% to 26.5% for the fine-grained strategy and 25.4% for the five-level quantized strategy, confirming that adapting the NoC frequency to the active packet length improves the effective use of the network. On the hardware side, the per-node area increases almost linearly from $2222.312, \mu\text{m}^2$ at $W = 16$ to $12375.404, \mu\text{m}^2$ at $W = 128$. This scaling is mainly dominated by the two data FIFOs in the NI and by the crossbar and output registers in the router, while the control-dominated blocks remain nearly constant. The preliminary physical design closes timing at approximately $800, \text{MHz}$ post-layout, lower than the synthesis-stage estimate because of interconnect parasitics and floorplan-induced wire delay, but still supportive of the overall feasibility of the architecture.

Chapter 6

Conclusions and Future Work

This chapter provides a conclusion and reflection on this thesis, and briefly outlines the future work plans.

6.1 Conclusions

This thesis presented the design, implementation, and characterization of a complete 5×5 torus NoC for a 25-node system. The work focused on developing a concrete hardware design point, starting from architectural choices and continuing through RTL implementation, simulation-based verification, synthesis, and preliminary physical design.

The proposed design combines a bidirectional torus topology, deterministic dimension-ordered routing, local injection control, and a single-cycle bufferless router with two spatial channels per direction. These choices were made to keep the design structurally simple while still providing a complete network architecture that could be implemented and evaluated as hardware. The whole system was designed, and a separate verification infrastructure was developed, forming a full RTL design rather than only an architectural model.

The evaluation showed that the implemented design behaved correctly in the tested deterministic and random permutation traffic cases. The self-checking testbench verified correct delivery across 1.25×10^6 packets with no observed flit-level errors. The measured static-frequency link utilization matched the analytical prediction, while dynamic NoC frequency scaling improved average utilization from 20.4% to 26.5% for the fine-grained strategy and to 25.4% for the five-level quantized strategy. The hardware results showed that the per-node area scales almost linearly with payload width, increasing from $2222.312, \mu\text{m}^2$ at $W = 16$ to $12375.404, \mu\text{m}^2$ at

$W = 128$, mainly due to the NI data FIFOs, router crossbar, and output registers. The preliminary physical design achieved post-layout timing closure at approximately 800, MHz.

Overall, this thesis provides a coherent baseline implementation of a bufferless torus NoC and reports its functional, utilization, area, and timing characteristics under the assumptions and test cases considered in this work. The result is not intended as a universal or optimized NoC solution, but as a complete and traceable design point that connects architectural decisions with concrete RTL and preliminary physical-design outcomes.

6.2 Limitations

The scope of this thesis is limited to permutation-style traffic, mainly shift permutations and random permutations. Therefore, the results do not directly predict performance for workloads with strong locality, hotspot traffic, multicast communication, or irregular application-specific communication patterns. In addition, dynamic NoC frequency scaling was evaluated as an architectural and simulation-level mechanism, not as a complete hardware frequency-control subsystem. Finally, the physical implementation was preliminary and was intended for characterization rather than sign-off. It does not include final timing closure, complete power analysis, design-for-test insertion, or tapeout-level verification.

6.3 Future work

Future work can extend this thesis in three directions. First, the NoC should be evaluated under a broader set of traffic patterns and real application traces, including localized, hotspot, multicast, and irregular communication, to determine how well the proposed architecture generalizes beyond permutation traffic. Second, the dynamic-frequency mechanism should be implemented as a complete hardware subsystem, including clock-generation, control, synchronization, and power-cost evaluation, so that its practical benefit can be measured more accurately. Third, the physical design should be taken further toward a sign-off-quality implementation, including full timing closure, post-layout power analysis, design-for-test support, and more detailed floorplanning of the tile and NoC regions. These extensions would provide a more complete assessment of the architecture as a practical on-chip interconnect.

References

[1] D. C. Jung, M. Ruttenberg, P. Gao, S. Davidson, D. Petrisko, K. Li, A. K. Kamath, L. Cheng, S. Xie, P. Pan *et al.*, "Scalable, programmable and dense: The hammerblade open-source risc-v manycore," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 770–784.

[2] J. Choquette, "Nvidia hopper h100 gpu: Scaling performance," *IEEE Micro*, vol. 43, no. 3, pp. 9–17, 2023.

[3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[4] S. Lie, "Cerebras architecture deep dive: First look inside the hardware/software co-design for deep learning," *Ieee Micro*, vol. 43, no. 3, pp. 18–30, 2023.

[5] W. J. Dally and B. Towles, "Route packets, not wires: on-chip inteconnection networks," in *Proceedings of the 38th annual design automation conference*, 2001, pp. 684–689.

[6] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *computer*, vol. 35, no. 1, pp. 70–78, 2002.

[7] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, pp. 1–es, 2006.

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

[9] A.-M. Rahmani, A. Afzali-Kusha, and M. Pedram, "A novel synthetic traffic pattern for power/performance analysis of network-on-chips using negative exponential distribution," *Journal of Low Power Electronics*, vol. 5, no. 3, pp. 396–405, 2009.

[10] J. H. Takala, T. Jarvinen, and H. T. Sorokin, "Conflict-free parallel memory access scheme for fft processors," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03.*, vol. 4. IEEE, 2003, pp. IV–IV.

[11] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-0998.html

[12] W. Cai, J. Jiang, L. Qin, J. Cui, S. Kim, and J. Huang, "Shortcut-connected expert parallelism for accelerating mixture-of-experts," *arXiv preprint arXiv:2404.05019*, 2024.

[13] C.-S. Chang, W.-J. Chen, and H.-Y. Huang, "Birkhoff-von neumann input buffered crossbar switches," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, vol. 3. IEEE, 2000, pp. 1614–1623.

[14] X. Zhao, S. Ma, Z. Wang, N. E. Jerger, and L. Eeckhout, "Cd-xbar: A converge-diverge crossbar network for high-performance gpus," *IEEE Transactions on Computers*, vol. 68, no. 9, pp. 1283–1296, 2019.

[15] S. Yue, L. Chen, D. Zhu, T. M. Pinkston, and M. Pedram, "Smart butterfly: reducing static power dissipation of network-on-chip with core-state-awareness," in *Proceedings of the 2014 international symposium on Low power electronics and design*, 2014, pp. 311–314.

[16] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: a cost-efficient topology for high-radix networks," in *Proceedings of the 34th annual international symposium on Computer architecture*, 2007, pp. 126–137.

[17] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 77–88, 2008.

[18] C. Clos, “A study of non-blocking switching networks,” *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[19] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo, “Designing application-specific networks on chips with floorplan information,” in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, 2006, pp. 355–362.

[20] Dally and Seitz, “Deadlock-free message routing in multiprocessor interconnection networks,” *IEEE Transactions on computers*, vol. 100, no. 5, pp. 547–553, 1987.

[21] L. G. Valiant and G. J. Brebner, “Universal schemes for parallel communication,” in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, 1981, pp. 263–277.

[22] 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.

[23] M. Moadeli, W. Vanderbauwhede, and A. Shahrabi, “A performance model of communication in the quarc noc,” in *2008 14th IEEE International Conference on Parallel and Distributed Systems*. IEEE, 2008, pp. 908–913.

[24] W. J. Dally *et al.*, “Virtual-channel flow control,” *IEEE Transactions on Parallel and Distributed systems*, vol. 3, no. 2, pp. 194–205, 1992.

[25] 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.

[26] H. Kasan and J. Kim, “Vvq: Virtualizing virtual channel for cost-efficient protocol deadlock avoidance,” in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 1072–1084.

[27] T. Fischer, M. Rogenmoser, T. Benz, F. K. Gürkaynak, and L. Benini, “Floonoc: A 645-gb/s/link 0.15-pj/b/hop open-source noc with wide physical links and end-to-end axi4 parallel multistream support,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 33, no. 4, pp. 1094–1107, 2025.

[28] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, vol. 2, 2004. doi: 10.1109/DATE.2004.1269001 pp. 890–895 Vol.2.

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

[30] S. Liu, Z. Lu, and A. Jantsch, "Highway in tdm nocs," in *Proceedings of the 9th International Symposium on Networks-on-Chip*, 2015, pp. 1–8.

[31] Z. Lu and A. Jantsch, "Tdm virtual-circuit configuration for network-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 8, pp. 1021–1034, 2008.

[32] S. Liu, A. Jantsch, and Z. Lu, "Parallel probe based dynamic connection setup in tdm nocs," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[33] ——, "Multics: Circuit switched noc with multiple sub-networks and sub-channels," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 423–434, 2015.

[34] J. Sparsø, H. J. Damsgaard, D. Katsamanis, and M. Schoeberl, "Comparing timed-division multiplexing and best-effort networks-on-chip," *Journal of Systems Architecture*, vol. 133, p. 102766, 2022.

[35] A. Leroy, P. Marchal, A. Shickova, F. Catthoor, F. Robert, and D. Verkest, "Spatial division multiplexing: a novel approach for guaranteed throughput on nocs," in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2005, pp. 81–86.

[36] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor, "Concepts and implementation of spatial division multiplexing for guaranteed throughput in networks-on-chip," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1182–1195, 2008.

[37] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 196–207.

[38] M. Hayenga, N. E. Jerger, and M. Lipasti, "Scarab: A single cycle adaptive routing and bufferless network," in *Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture*, 2009, pp. 244–254.

[39] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 196–207.

[40] C. Fallin, C. Craik, and O. Mutlu, "Chipper: A low-complexity bufferless deflection router," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 2011, pp. 144–155.

[41] C. Fallin, G. Nazario, X. Yu, K. Chang, R. Ausavarungnirun, and O. Mutlu, "Minbd: Minimally-buffered deflection routing for energy-efficient interconnect," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*. IEEE, 2012, pp. 1–10.

[42] R. Ausavarungnirun, C. Fallin, X. Yu, K. K.-W. Chang, G. Nazario, R. Das, G. H. Loh, and O. Mutlu, "Design and evaluation of hierarchical rings with deflection routing," in *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*. IEEE, 2014, pp. 230–237.

[43] ——, "A case for hierarchical rings with deflection routing: An energy-efficient on-chip communication substrate," *Parallel Computing*, vol. 54, pp. 29–45, 2016.

[44] T. Wang, F. Feng, S. Xiang, Q. Li, and J. Xia, "Application defined on-chip networks for heterogeneous chiplets: An implementation perspective," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 1198–1210.

[45] H. Farrokhbakht, P. V. Gratz, T. Krishna, J. San Miguel, and N. E. Jerger, "Stay in your lane: A noc with low-overhead multi-packet bypassing," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 957–970.

[46] Z. Jin, C. Rocca, J. Kim, H. Kasan, M. Rhu, A. Bakhoda, T. M. Aamodt, and J. Kim, "Uncovering real gpu noc characteristics: Implications on interconnect architecture," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 885–898.

[47] S. Chen, S. Pal, and R. Kumar, "Waferscale network switches," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 215–229.

[48] K. Salimifard and S. Bigharaz, "The multicommodity network flow problem: state of the art classification, applications, and solution methods," *Operational Research*, vol. 22, no. 1, pp. 1–47, 2022.

[49] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto noc architectures," in *Proceedings design, automation and test in Europe conference and exhibition*, vol. 2. IEEE, 2004, pp. 896–901.

[50] 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," *IEEE transactions on parallel and distributed systems*, vol. 16, no. 2, pp. 113–129, 2005.

TRITA-EECS-EX-2026:391
Stockholm, Sweden 2026

www.kth.se