DEMOCRITUS
UNIVERSITY
OF THRACE

High Performance Networks-on-Chip

PhD Thesis

Anastasios Psarras

July 11, 2017

Advisor: Assistant Prof. Giorgos Dimitrakopoulos

Electrical & Computer Engineering Dept.
Democritus University of Thrace, Xanthi, Greece

Abstract

Over the last two decades, we have witnessed a fundamental paradigm shift in digital system design: the transition to the multicore realm. Naturally, the multicore domain has elevated the criticality of the on-chip interconnection fabric, which is now tasked with satisfying amplified communication demands. Owing to their scalability attributes, Networks-on-Chip (NoC) have established their position as the de facto communication medium in multicore systems. To sustain system scalability into the many-core domain (with potentially hundreds of cores), it is imperative that the NoC's hardware cost is minimized, while not sacrificing network performance.

To this end, we propose three alternative architectures that can significantly improve the performance of NoCs, or lead to an overall lower power consumption.

The first one is a pipelined router architecture, called ShortPath, that parallelizes – for the first time – the allocation steps involved in the operation of a VC-based router without resorting to speculation. Most importantly, ShortPath is augmented with an always-productive pipeline bypassing mechanism, which skips all stages without contention, and "fast-forwards" the flits to the first encountered point of contention.

The other two approaches exploit fast link traversal, after appropriate wire engineering, to rapidly transfer flits between adjacent routers (connected with links of reasonable, short-to-medium length of up to a few millimeters) in half a clock cycle. Under this clocking principle, two design alternatives are explored, which allow for half-cycle and Double-Data-Rate (DDR) link traversal. The proposed approaches can markedly increase network performance, or decrease the area/power cost of the NoC. Although not obvious at first glance, half-cycle link traversal opens up new possibilities for reducing wire capacitance. By harnessing these opportunities, RapidLink renders the half-cycle-delay requirement easier to achieve and potentially extends half-cycle traversal capabilities to longer links. To tackle longer links, novel DDR dual-stream elastic buffers are proposed for pipelining the links while still following DDR flow control.

As multi/many-core architectures evolve, the demands on the NoC are amplified. In addition to high performance and physical scalability, the NoC is increasingly required to also provide specialized functionality, such as network virtualization, flow isolation, and quality-of-service guarantees. Although traditional architectures supporting Virtual Channels (VCs) offer the resources for flow partitioning and isolation, an adversarial workload can still interfere and degrade the performance of other workloads that are active in a different set of VCs.

Motivated by this aspect, we present PhaseNoC, a truly non interfering VC-based architecture that adopts time-division multiplexing at the VC level. Distinct flows, or application domains, mapped to disjoint sets of VCs are isolated, both inside the router's pipeline, and at the network level. Any latency overhead is minimized by appropriate scheduling of flows in separate phases of operation, irrespective of the chosen topology. When strict isolation is not required, the proposed architecture can employ opportunistic bandwidth stealing to further reduce packet latency.

Acknowledgements

First of all, I would like to thank my advisor, Giorgos Dimitrakopoulos, for giving me the chance to work with him, and be the first to complete doctoral studies under his guidance. His expertise allowed me to face the technical challenges, while his constant and irreducible enthusiasm has always been a driving force that kept me going despite the difficulties. I am grateful to Giorgos for all the professional and research opportunities he offerred me, for his insightful advice in good and bad times, and, most importantly, for his fair professional attitude.

I would also like to thank the members of my advisory and defence comittee, Dionisios Pnevmatikatos, Giorgos Syrakoulis, Vassilis Paliouras and Manolis Katevenis for evaluating and reviewing my work, and for providing insightful comments that helped to improve this thesis. I would especially like to thank Chrysostomos Nicopoulos for our long-lasting and productive collaboration, for his assistance in my research efforts and his priceless contribution in my published works. I also owe a special thanks to Ioannis Karafyllidis for the appreciation and trust he has showed on me, and for providing me with invaluable and open-minded advice.

I would also like to express my gratitude to the rest of the people we have collaborated throughout these years. My colleague, Ioannis Seitanidis, and Junghee Lee, from the University of Texas at San Antonio, for investing time and effort in developing our simulation frameworks and collecting experimental results. Also, to Nick Papanikolaou, for giving me the chance to work on his research projects and, finally, Todor Mladenov and Helmut Reinig, for trusting me to work freely and autonomously during my intership at Intel Mobile in Munich, Germany.

I also need to thank the rest of the people that made a great company during short coffee breaks, or longer raki and beer nights, helping me relax and release the pressure, Dimitris K., Giorgos B., Chris, Maria, Kostis, Greg, Fivos, Mihai, David and Majid.

A sincere “thank you” goes to my parents, Yannis and Aphrodite, and to the rest of the family, Chris, Kiki and Irene for their unconditional support and constant encouragement.

This PhD thesis is dedicated to Clara, who has stood patiently by my side at all times; I would not make it to this point without her love and support.

Contents

Chapter 1

Introduction

Over the past two decades, the world has witnessed the unprecedented proliferation of information technology, which has permeated all facets of our lives. In fact, Information and Communication Technology (ICT) has been the key enabler of nearly all efforts to advance science and improve humankind's quality of life. Innovations in ICT have been primarily driven by two forces: (a) semiconductor technology and (b) computer architecture. The everlasting miniaturization of semiconductor technology feature sizes has resulted in exploding transistor integration densities. More importantly, the elevated numbers of transistors per chip are accommodated at approximately the same power levels and cost. Complementary to technological improvements, innovations in computer/system architecture have been utilizing the increased transistor budgets very effectively, thereby managing to dramatically improve the performance of microprocessors. The synergistic impact of semiconductor technology and computer architecture has precipitated exponential performance improvements at nearly constant hardware cost.

We have reached a point where transistor integration capacity will continue with scaling, though with limited performance and power benefit [23]. Computer architects reacted to this challenge with multicore architectures that help close this gap. Multicore refers to a single chip containing multiple visibly distinct processing engines, each with independent control. The additional chip area offered by each new technology generation at a constant chip area is filled with more cores. The first systems developed followed a homogeneous architecture, while recent approaches, move gradually to heterogeneous systems that look like complex platform System-on-Chips (SoCs). These platforms integrate in the same chip some latency-optimized cores, many throughput-optimized cores (like GPUs), and some specialized units, including the associated memory hierarchies, thus covering the needs of many application domains. Programming such heterogeneous systems in a unified manner is still an open challenge. Nevertheless, any revolutionary development in heterogeneous systems programming should rely on a solid computation and communication infrastructure that will aid and not limit system-wide improvements.

Scalable interconnect architectures form the solid base on top of which heterogeneous computing platforms and their unifying programming environments will be developed [2]. After all, parallelism is all about cooperation that cannot be achieved without the efficient communication offered by the interconnect [9]. The interconnect architecture should be as high-performance as its connecting nodes, thus enabling the expected exponential growth in system concurrency. The interconnect implements the physical and logical medium for any kind of data transfer and its latency, bandwidth and energy efficiency directly affects overall system performance. Interconnect design is a multidimensional problem involving hardware and software components such as network interfaces, switches, topologies, routing algorithms and communication library APIs.

Modern heterogeneous multiprocessing systems have adopted a Network-on-chip (NoC) technology that brings interconnect architectures inside the chip [54, 29]. The NoC paradigm tries to find a scalable solution to the tough integration challenge of modern SoCs, by applying at the silicon-chip level well-established networking principles, after suitably adapting them to the silicon-chip characteristics and to application demands. This approach was originally adopted to tackle the physical integration complexity, clocking scalability, timing closure and verification problems of state-of-the-art SoCs. While the seminal idea of applying networking technology to address the chip-level interconnect problem has been shown to be adequate for current systems, the complexity of future computing platforms demands new architectures that go beyond physical-related requirements and equally participate in delivering high-performance, quality of service, dynamic adaptivity at the minimum energy and area overhead.

Figure 1.1 illustrates a tile-based 16-core Chip Multi-Processor (CMP), interconnected through a 4×4 2D mesh on-chip network [6, 34, 121]. Each tile consists of one or more CPU cores, private L1 instruction and data caches and a slice of L2 cache (more cache levels are possible). A Network Interface (NI) unit attached to the tile acts as a gateway to the local NoC router, converting messages into packets and back. Packets are then routed to their destination one hop at a time, switching and contending with other flows in network routers, and traversing the physical distance through the inter-router links.

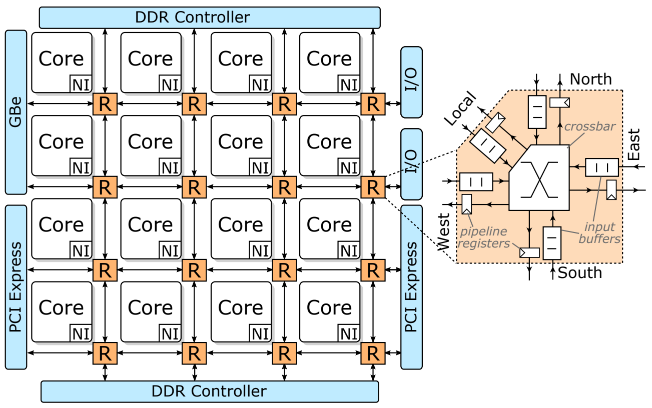


Figure 1.1: A typical multi-core system connected with a Network-on-Chip.

1.1 Network Transactions & Interfaces

Messages exchanged between the system's cores are parts of transactions, that describe a memory accesses (write or read) on an address range. Transactions are triggered by executable software code and their effect depends on the system organization. For example, a write transaction may be executing a store operation from a CPU to the memory, or a switch of the CPU's power mode, through a write on a configuration register.

A typical transaction is initiated with a *request* message, containing the transaction header (address, access type, ordering requirements etc.) and, in case of a write, the payload (data). After the request is executed at the destination core, a *response* is generated, acknowledging the operation, and sending back the requested payload, in case of a read. Transactions are presented by the IP cores to the NoC through a strictly-defined socket protocol, such as AXI [7] or OCP [4]. Socket protocols are limited in defining the rules over a Master-Slave interface, in which, only the Master is allowed to initiate a transaction; the Slave can only respond to Master requests. The Master-Slave abstraction effectively decouples transaction- from network-level semantics, allowing designers to customize the NoC operation according to the system needs.

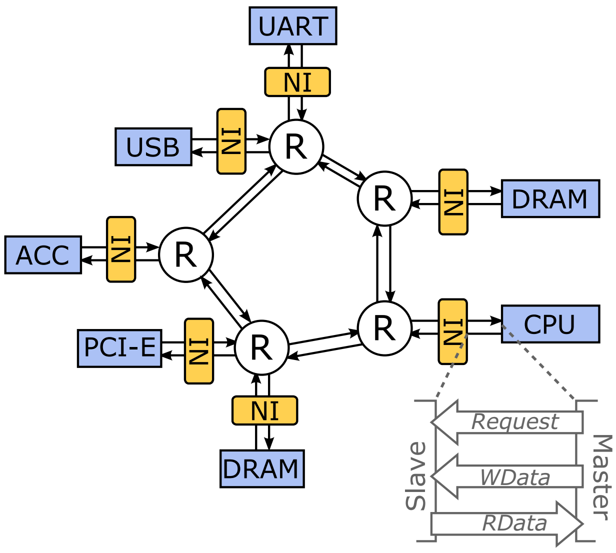


Figure 1.2: A Network-on-Chip that connects System IPs through Network Interface modules.

Interfacing between the internal NoC protocols and the socket protocols of the IP cores is performed through Network Interfaces (NIs) [113, 41, 133] at the NoC periphery, as shown in the example system of Figure 1.2. Each NI exposes a single complementary interface towards the IP core [110, 136], effectively concealing the NoC's complexity from the IP. For instance, the Master CPU of Figure 1.2 connects to one of the NoC's Slave ports, while the memory Slave connects to a Master NoC port. Once the Slave NI receives the request generated by the Master and converts it into a packet. Once the packet reaches its destination, it is converted back into a request by the Master NI, and presented to the Slave interface of the destination IP. The IP's response is then converted back to a packet by the Master NI and injected to the network. Once the packet reaches its destination, it is converted back to a response by the Slave NI, completing the transaction cycle.

A single message is converted to one or more packets, and each packet is divided into multiple flits [113][1]. The head (i.e. the first) flit of the packet carries the network header, which contains routing and flow-control in-

[1]A flit may be further divided into *phits*, the size of which represents the minimum flit width that can exist in the network.

formation that allows the packet to follow its route in the network, and to convert it back to a message at the NI of the destination node. It contains the source and destination ID, the message type (e.g. write/read and request/response), ordering and Quality of Service (QoS) information etc. Multiple body flits carry the message payload, with a tail flit marking the end of the packet. In case the packet can fit in one flit, a single flit marks both the start and the end of a packet.

An example of a packetized write request is shown in Figure 1.3. At the transaction level (Figure 1.3(a)), the write request follows memory access semantics (write/read, burst, address, data). The NI appends a network header containing routing and flow-control information to form the packet (Figure 1.3(b)), which is "chopped" into multiple flits to be injected in the network (Figure 1.3(c)).

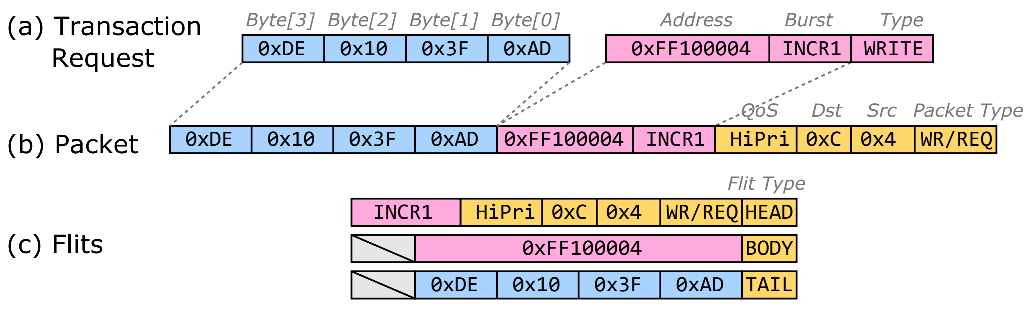


Figure 1.3: An example of a (a) write transaction, (b) converted to a network packet, (c) that consists of multiple flits that will be injected to the network.

Choosing a packetization scheme and flit width presents a critical decision with a significant performance-cost trade-off: wide flits can lead to smaller packets and thus, less serialization latency, but they increase area due to wider buffers and datapath logic. In the SoC domain, designers may implement asymmetric link (flit) widths throughout the network [130, 78], to make the most of this performance-area trade-off, depending on the bandwidth requirements of each network node. E.g. a wider link could be used to compensate for a slow clock frequency, or to avoid limiting bandwidth on an IP core with a wide data bus. In that case, de/serialization units are employed to up/downsize to a wider/narrower flit [64]. In CMPs, where message types are less diverse, with fixed cacheline-sized payloads, a universal link width is preferred [126, 81].

Packets that enter the network may either be consumed by the destination NI as shown in Figure 1.4(a), or they may cause the generation

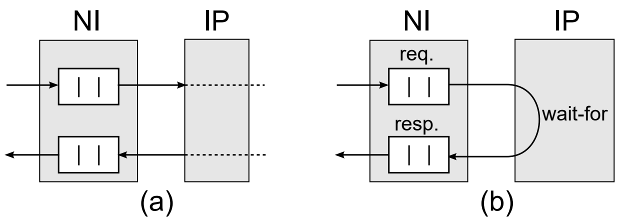


Figure 1.4: (a) Illustration of the consumption assumption, made by deadlock-free routing algorithms does not apply in systems where a (b) request-response dependency is introduced by Slave IPs.

of a new packet that will enter the network with the goal to be delivered some cycles later. The second case is the most common one in modern transaction-level protocols where a master's request packet is followed by the slave's reply or an acknoweledgement packet that must return to its owner (requesting master). This case introduces a request-response dependency, as shown in Figure 1.4(b): in order for the Slave IP to consume a request, the response buffer at the NI must have free space available. Since the Slave cannot guarantee the sinking of requests in finite time, extra dependencies are added to the network [56, 94], and a cyclic deadlock is possible, where no more requests can be consumed nor any replies can be transmitted.

Figure 1.5 illustrates an example of such a deadlock possibility. Figure 1.5(a) shows a 4-input/4-output NoC that is free of cyclic dependencies. Given that the consumption assumption is satisfied at the network's endpoints, no deadlocks are possible. However, due to the request-response dependency added by the connected Slaves, a hazardous cycle is indeed formed (highlighted in red in Figure 1.5(b)). If Masters 1 and 2 initiate a request simultaneously to Slaves 1 and 2 respectively, the cyclic dependency will be activated and the network *will* be led to a deadlock.

In order to guarantee deadlock freedom, the consumption assumption has to be fulfilled, either by breaking the cycle completely, or by making sure that it will never be activated [56]. The latter can be achieved by over-sizing buffers, so that packets are never blocked from allocating buffer space. Another method involves employing end-to-end flow control, guaranteeing that packets will find buffer space available at their destination [40, 110]. To avoid the increased buffering requirements of such methods, the dependency cycle can be explicitly broken, by employing multiple Physical [90] or Virtual Networks [134].

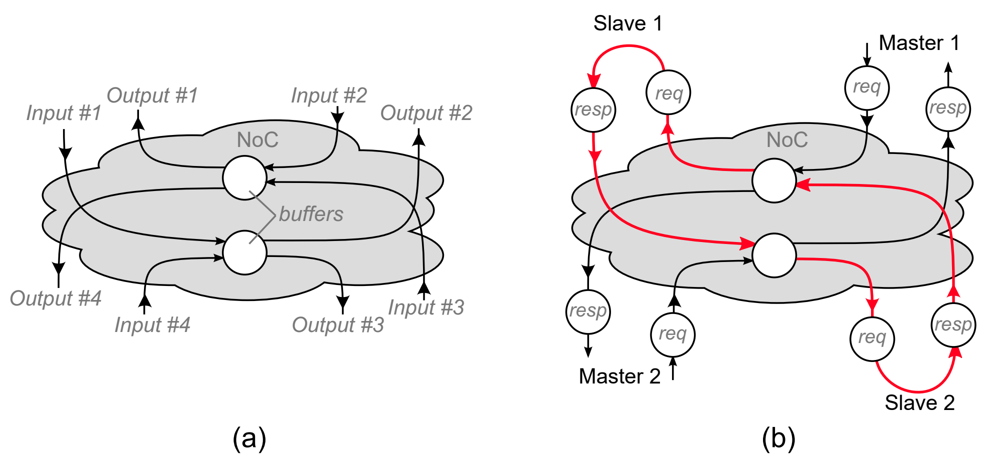


Figure 1.5: (a) An initially deadlock-free NoC, without cyclic dependencies (b) can become deadlock-prone when request-response dependencies are added at its endpoints.

1.2 Network Organization

The way that each packet entering the network from a NI will be delivered to its final destination NI depends on four main network parameters/attributes that all of them determine the network's performance in terms of latency and throughput, as well as hardware implementation cost:

- **Topology:** The way that the network nodes are connected. The connectivity refers to the aggregation of network interfaces to network routers and the connectivity of the network routers using network's links. The network topology defines the paths on which packets can move. The router is the hardware module placed at the crossroads of the network and should forward to the correct output all traffic that arrives at its inputs.

- **Routing:** The routing function selects the paths that a packet must take from its source to its destination. Routing should enable connectivity among all nodes and correct packet delivery, while it should also target load balancing by distributing evenly the incoming traffic to the network's links.

- **Flow control:** The flow control scheme governs how routers communicate with each other and it determines when packets or flits can be forwarded from one router to the next. Each packet moves in the network as a unified entity, crossing links, and switching in the network routers, in a hop by hop manner until it reaches its destination. Each input/output port of the router that is connected to the network's links should be independently flow-controlled providing lossless operation and high communication throughput.

- **Router microarchitecture:** The router is the active hardware module that accepts, buffers, and forwards appropriately the incoming packets. The internal organization of the NoC's router is crucial in achieving high clock frequencies, high throughput and low latency operation, without exceeding the overall NoC's power budget.

1.2.1 Topology

The way that the network nodes are connected defines the network's topology. Figure 1.6 shows three different topologies used to connect 16 homogeneous tiled cores: (a) a ring, (b) a 2D Mesh and (c) a flattened butterfly. Since the NoC has to fit in the space left after the rest of the IPs are placed, the choice of the topology structure is primarily defined by the physical layout of the existing IP cores in the chip [59, 28]. In homogeneous tiled CMPs, the regularity of the cores' layout allows the use of regular and homogeneous topologies, with minimal wiring overhead, such as rings and meshes. E.g. Intel employs a ring topology for the 4-core Skylake CMP [6], and a mesh for the 36-core Knight's Landing architecture [121].

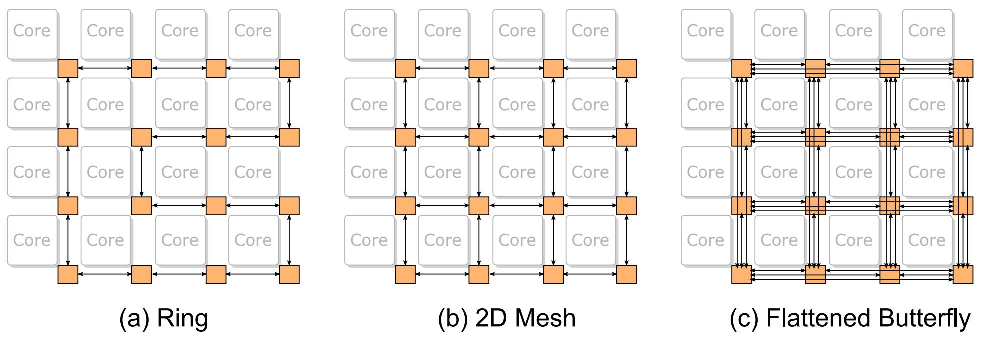


Figure 1.6: Three different 16-node topologies: (a) a ring, (b) a 2D mesh and (c) a flattened butterfly.

Topology determines the bounds of both packet delivery latency and the throughput of the network as a whole. Minimizing the packet latency that each packet experiences when travelling in the network, while maximizing communication throughput, is one of the main goals of high-performance NoCs. Fast packet delivery directly translates to faster execution times, while higher throughput increases the amount of data that can be communicated and processed in parallel. Throughput relates to the topology's bisection bandwidth as it bounds the amount of data that can be communicated simultaneously [31]. Packet latency depends on the network diameter, i.e., the minimum distance between the two most distant nodes. A packet with a length of L flits, that has to travel H hops to reach its destination, will be delivered in $T_h + T_S = H(t_R + t_L) + L - 1$ cycles. Header latency (T_h) is the time that the head of the packet needs to reach its destination and is equal to the latency of each hop in the network, i.e. the cycles spent on each router (t_R) and on each link (t_L). Serialization latency (T_S) is the number of cycles for the rest of the packet, consisting of $L - 1$ flits, to arrive after the head.

The network topology determines also the hardware cost, since implementation gets more complex with more diverse topologies. The number of router input/output ports (their radix), defines the router implementation complexity and thus, its maximum operating frequency. For example, mesh routers in Figure 1.6(b) needs 5 input/output ports (4 directions plus a local port to the NI), and thus, they can operate on a higher clock frequency compared to the 7-port routers used in the butterfly network (Figure 1.6(c)). Therefore, although the number of hops in the butterfly network is lower than the mesh, each hop will cost more in absolute time, due to increased clock period. Moreover, the topology affects the physical characteristics of the links. Longer wires need repeaters to satisfy tight timing constraints, increasing, in turn, their power consumption. If repeaters do not suffice to satisfy timing constraints, the link may have to be pipelined, leading to increased round trip times, and thus, more buffering requirements to achieve full-throughput operation on long inter-router channels.

1.2.2 Routing

Network topology defines the physical organization of the network composed by the nodes, and thus the available paths between all the nodes. The routing algorithm is responsible for deciding which path the packet should follow to be effectively routed from its source to its destination. The main purpose of an efficient routing algorithm is to be simple enough so that it can be efficiently implemented either at the NIs or at the routers following a distributed organization. Also, the routing algorithm should utilize all network links as uniformly as possibly trying to avoid traffic concentration at certain parts of the network that limits throughput and leaves other parts of the network underutilized. The third, and possibly, the most crucial role of the routing algorithm is to be deadlock free and to guarantee the correct delivery of messages among all source-destination pairs.

A deadlock situation can occur in a NoC when a cyclic dependency is formed among packets requesting an occupied resource of the network. The most common deadlock conditions arise when the requested resource involves the buffer space distributed inside the network. Consider the case of Figure 1.7(a), that demonstrates a deadlock condition between 4 packets crossing 4 2D mesh nodes. Packet A wants to turn south at node 3, but it is blocked by Packet B that has already allocated exclusive access to channel c_{13}. In turn, Packet B requests turning east at node 3 but is blocked by Packet C that has already allocated channel c_{34}. At the same time, Packet C requires a north turn but is stopped by Packet D. Finally, Packet D needs to turn west but is blocked by Packet A that in a previous cycle has won access to channel c_{42}. A cyclic dependency has been formed, and all packets are unable to proceed.

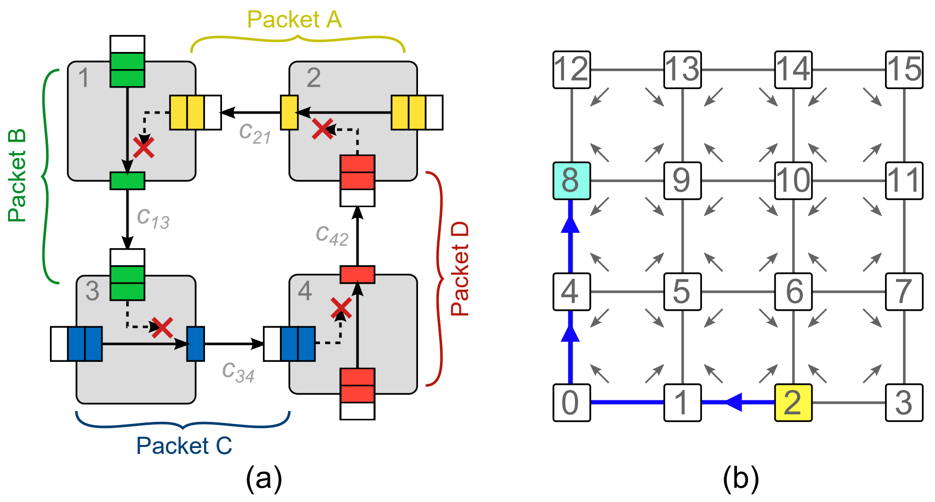


Figure 1.7: (a) Cyclic paths formed inside the network lead to cyclic resource dependencies that can lead to a deadlock. (b) Deadlock-freedom on the XY routing algorithm is guaranteed by restricting all possible turns from the Y to X dimension. A packet moving from node 2 to node 8 will first have to follow the X dimension, turning only whenit reahes the Y coordinate of its destination.

To avoid such deadlocks the routing algorithm must guarantee that no cyclic dependencies are possible within the network, by limiting the paths a packet can follow to reach its destination. This path limiting process is actually a network-level planning on the allowed turns that a packet can take when traversing the network links and routers. Figure 1.7(b) an example of the XY deadlock-free routing for 2D meshes, where the arrows on the network constitute carefully-designed turn prohibits. A packet routed from node 2 to node 8 is only allowed to turn once during its route, from direction X to Y. With XY routing, the deadlock scenario of Figure 1.7(a) would not have occured, since Packets B and D would never turn from the Y dimension.

1.2.3 Flow Control

Flow control is the mechanism that describes how a packet or flits of packet are transferred between two endpoints, either across NIs (end-to-end flow control) or across routers (link-level flow control) providing lossless operation and high communication throughput.

Links are the physical medium that allow data to traverse physical distances within the chip. They consist of a data bus, that transfers flit data, and a pair of opposite-direction signals, used to control the transmission. Exchange of link-level flow-control information on both directions is necessary so that the sender is always aware of the transmission status. Packet dropping, used extensively in large-scale networks, are prohibitive at the on-chip environment, due to their increased buffering requirements, and the design implications they incur (e.g. packet fragmenting, out-of-order packet arrivals etc.). Therefore, we strictly focus on lossless link-level flow-control policies.

At any link of the network, the packet must first allocate necessary downstream buffer space before it can move forward. With packet-buffer flow-control policies, like cut-through and store-and-forward the head of the packet must first allocate space for the whole packet before it can move forward to the next buffer. In contrast, with *wormhole* flow control, channels and buffers are allocated at the flit level: every flit of a packet can move forward to a downstream buffer, as long as there is one free slot to accommodate it. At any case, router allocation mechanisms, make sure that the packet will still move as a unified entity, without mixing with other packets in the network buffers.

Wormhole flow-control has minimal buffering requirements and avoids the latency overhead of packet-buffer policies, which are rarely used. However, the serialization of different packets in the network buffers introduces inter-flow blocking. This blocking introduces deadlock-prone dependencies between message classes that need to remain independent (e.g. requests and responses), and is responsible for the Head-of-Line blocking (HoL) effect, that degrades network throughput.

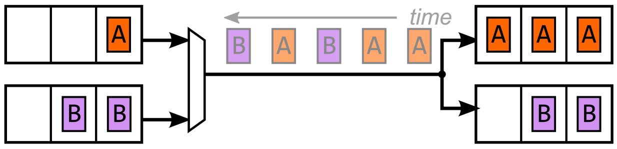


Figure 1.8: With a Virtual Channel flow-control-based scheme, each buffer point consists of multiple parallel buffers, and their intermediate link is shared in time.

With Virtual Channels (VCs), the link can be shared by multiple packet flows. As shown in Figure 1.8, for each VC, a separate buffer is required at each buffering point. Before a packet can move forward, it must first allocate a downstream VC and block any other packet from accessing it. Then, each flit of the packet tries to acquire a timeslot, after arbitrating with the rest of VCs, for actually crossing the link.

Virtual Channels are an integral part of most NoC architectures, since they contribute to performance enhancement, and they facilitate network traffic separation. The latter attribute enables, in turn, network virtualization, traffic isolation [107], and quality-of-service provisioning [79]. Moreover, VCs can be used for the definition of Virtual Networks (VNs), which are used to avoid deadlocks. Packets belonging to a VN complete their entire trip in the network by traversing the same VN. In-flight transitions from one VN to another are either prohibited, or done with very restrictive rules, which are used to guarantee deadlock freedom in the cases of: (a) separating request/reply traffic [56] and/or other cache coherence message classes [87], and (b) supporting adaptive routing [38].

Buffer availability at the other side of the link can be performed with two methods: (a) credit-based or (b) elastic link-level flow control.

In **credit-based** flow-control, the sender is keeps track of the free slots at the receiver's buffers using credit counters, and transmits flits only when the receiver has at least one free buffer slot to accommodate it. Whenever a flit is transmitted, the sender decrements its counter, in order to reflect the receiver's new buffer state and asserts the forward `valid` signal. When the receiver removes a flit from its buffer, a credit is sent back to the sender through the assertion of the `credit` signal, indicating that a buffer slot has been freed. When the sender receives the credit, it increments the credit counter value, effectively reflecting the new state of the receiver's buffer.

Achieving uninterrupted, full-throughput transmission over a credit-controlled link, requires that the receiver has enough buffer slots to cover the Round-Trip-Time (RTT). The RTT equals number of cycles elapsed from the time the Sender puts a flit on the link, until the receiver extracts it from its buffer and the associated credit reaches the sender. In the pipelined link of Figure 1.9, there are two register stages in the forward path (`valid/data`) and two more in the backward path (`credit`). Therefore, the receiver needs at least four buffer slots, so that the sender is never throttled, unless, of course, the receiver stops extracting flits from its buffer due to contention. If the receiver owns less than four slots, no data will be lost; only the link's throughput will be degraded.

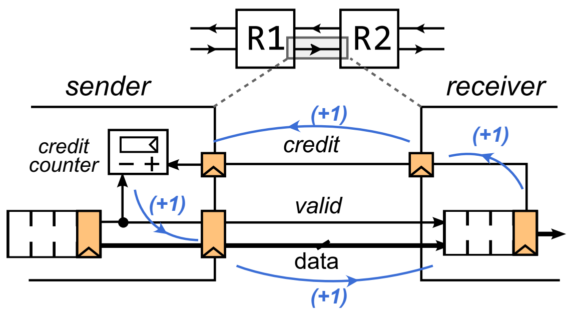


Figure 1.9: The structure of a credit-based flow-controlled link between two adjacent routers. The sender keeps track of the receiver's free buffer slots (credits) using a credit counter and is responsible for transmitting data only when the receiver has buffer slots available.

Credit-based flow control is the most widely adopted flow control method. Extending the protocol to support VCs only requires multiplying the senders credit counters, and adding an extra backward signal from the receiver that indicates the VC to which the returning credit belongs. Link pipelining can be achieved using simple pipeline registers without the need for special buffer structures. Most importantly, credit-based flow control offers the ability to implement complex buffer management schemes, allowing fine-grained bandwidth allocation to the various flows, with minimum overhead [115, 17].

With **elastic**, or "ready/valid" flow control, the sender has no information on the exact buffer status of the receiver and each transmission must be acknowledged [25]. A transmission is initiated by the sender with the assertion of the forward valid signal. In the following clock edge, the sender checks the receivers ready signal to conclude on the transmission status. If ready is asserted, then the transmission was successful; otherwise, the sender must keep the data in a local buffer and retry in the following clock cycle. Due to its intuitive operation and modular structure, elastic channels are preferred in most high-performance socket protocols [7, 4].

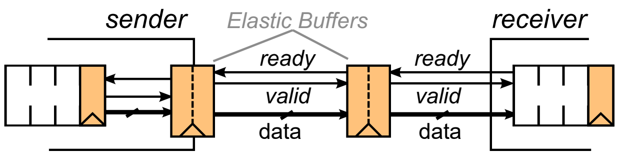


Figure 1.10: The structure of an elastic flow-controlled link between two adjacent routers.

An elastic flow-controlled link can be pipelined using Elastic Buffers (EBs) that provide consistent ready/valid interfaces, as shown in Figure 1.10. At least two buffer slots are required in order for an EB to completely isolate the timing paths on both forward (`valid/data`) and backward (`ready`) direction signals [61]. In wormhole links, EBs can be at least as buffer efficient as a credit-based link [36]. However, elastic protocols lack efficient Virtual Channel support, exhibiting deadlock hazards [70], increased buffering requirements [116] and lack of mechanisms for fine-grained buffer allocation.

1.2.4 Router organization

While link-level flow control enables lossless operation across a sender and a receiver in a one-to-one connection, and arbitration and multiplexing enables sharing a link by many peers, real network involve more complex switching cases that involve many-to-many connections. Each router should concurrently support all input-output permutations and solve the contention to all outputs at once respecting also the flow-control policy at the output links. Establishing a path between any source and destination of a complex network topology is a matter of the routing algorithm that is either implemented completely at the NIs or by the routers in a step-by-step and distributed manner.

When only one input requests a specific output, the router should connect the corresponding input with the designated output. When two or more inputs compete for gaining access to the same output in the same cycle the router is responsible for resolving the contention. This means that only one input will gain access to the output port. The flits of the input that lost stay it the input buffer of the current router and retry in the next cycle.

Despite a multitude of proposed NoC implementations and incarnations, the functionality expected from a VC-based router's microarchitecture remains the same: coordinate and direct the flow of packets from the inputs to the outputs of the router.

Arriving packets are written into the input VC buffers of the router. In the following cycles, they must find their way to the proper output port, after going through several allocation steps [36]. When a packet arrives in a router, it needs to find its output destination port via routing computation (RC). Each packet then has to choose a VC at the input of the next router, before leaving the current router (known as an "output VC"). Matching input VCs to output VCs is performed by the VC Allocator (VA). Allowing packets to change VC in-flight can be employed when the routing algorithm does not impose any VC restrictions (e.g., XY routing does not even require the presence of VCs). However, if the routing algorithm and/or the upper-layer protocol (e.g., cache coherence) place specific restrictions on the use of VCs, then arbitrary in-flight VC changes are prohibited, because they may lead to deadlocks. In the presence of VC restrictions, the VC allocator will enforce all rules during VC allocation to ensure deadlock freedom. The flits that own an output VC arbitrate for accessing their output port. If a flit wins this stage – called Switch Allocation (SA) and organized in local and global arbitration steps, SA1 and SA2 – it will traverse the crossbar (Switch Traversal, ST), and then it will move to the output link (Link Traversal, LT) towards the next router.

When the network does not support Virtual Channels, wormhole routers are used instead, which are significantly simplified [36]. In wormhole routers, no VA stage is required, and the allocation problem is solved using a single-step SA unit, that requires one $N:1$ arbiter per output port. As a result, wormhole routers can achieve very low-cost implementations, and can achieve faster clock frequencies than VC-based ones [134, 117]. However, wormhole routers have no inherent support for multiple message classes (e.g. request/response), and thus, they need to be replicated to form Physical Networks, increasing the network's wiring congestion. Still, due to wormhole routers' simplicity and low cost, physical networks have been employed in a variety of silicon-proven CMPs [60, 96, 111, 131].

1.3 Thesis Contribution

As the number processing cores integrated on CMPs and SoCs continues to grow, addressing the on-chip communication demands with an efficient fabric, becomes a major challenge. Networks-on-Chip have been established as the dominant communication medium, due to their modular approach, their physical scalability and ease of integration.

To sustain future multi-core scaling without becoming the system's bottleneck, NoCs must provide high communication throughput and low latency transfers, as well as specialized functionality, such as network virtualization, flow isolation, and QoS guarantees. These features must be provided under efficient hardware implementations that satisfy tight timing, area and power constraints.

Most of those challenges have been successfully satisfied by the research outcome presented in this thesis that is summarized to the following contributions:

- A high speed pipelined router architecture that employs novel pipeline bypass techniques. ShortPath routers [108] achieve high operating frequency by parallelizing as much as possible – and without resorting to speculation – the allocation steps involved in the operation of a VC-based router. ShortPath is augmented with a fine-grained pipeline bypassing mechanism, which skips all stages without contention and "fast-forwards" the flits to the first point of contention. In this way, proposed routers enjoy the hardware benefits of pipelining, without suffering from increased packet latency.
- A novel network design methodology that guarantees flow isolation, by adopting Time Division Multiplexing (TDM) at the VC level. The proposed PhaseNoC architecture provides secure-grade flow isolation under a low-cost hardware implementation, while still improving the network performance of existing solutions that can provide similar services [107]. PhaseNoC's performance is significantly improved, when strict, secure-grade isolation features are not neccessary, with our proposed Opportunistic Bandwidth Stealing mechanism [105].
- A low-power technique for rapidly transferring flits over network links in half a clock cycle [104]. Half-cycle link traversals are facilitated by the inherent asymmetry between the delay of a typical router and a network link in tiled CMPs, improves packet latency,

while simulatenously offering significant reductions in: (a) link power, irrespective of the data switching profile, and (b) buffer power, through buffer-size reduction.

- The RapidLink architecture, that extends the half-cycle link traversal concept to achieve high-throughput, Double-Data-Rate (DDR) link transmissions [106]. With our proposed novel DDR Elastic Buffers, links can be pipelined in a plug-and-play manner, while preserving their DDR characteristic. In this way, RapidLink removes the link timing constraints, while preserving the throughput benefits of DDR operation.
- A complete set of parameterized and optimized RTL implementations covering all kinds of router organizations that helped us explore several microarchitecture alternatives and quantify the benefits of the proposed techniques. We plan to release publicly the developed framework for open source use.

1.4 Evaluation Methodology

For all the designs and methods proposed in this thesis, a common design flow was followed, along with a consistent evaluation methodology. All proposed methods were evaluated and compared against state-of-the-art solutions, both in terms of network performance and hardware efficiency. In all cases, we followed well-accepted methodologies of the NoC research community, using EDA tools broadly used in the industry.

1.4.1 Network & Full-System Simulations

Network performance was measured using a cycle-accurate simulator, developed in C++ with the aid of the SystemC library, used for describing hardware in a high-level abstraction [10]. High-level SystemC models were preferred – instead of more detailed ones, such as RTL descriptions – due to their fast development cycle. These high-level models allow for early performance assessment, long before the microarchitecture of the design has been finalized. At all cases, the accuracy of the models was cross-checked for consistency with the equivalent results of the actual hardware implementations.

Performance assessment was mainly performed on the two critical network characteristics, latency and throughput, measured under various

Table 1.1: System parameters for the full-system simulations.

Processor	64 in-order x86 cores @ 1 GHz in a tiled CMP architecture
L1 caches	Private, separate 32 KB I & D, 4-way set associative, 2-cycle latency, 64 B cache-line
L2 cache	Shared NUCA LLC, 4-way set associative, 16 MB total (64 cores×256 KB slice/core), 10-cycle latency, 64 B cache-line
Coherence	MOESI directory-based cache coherence protocol
Main memory	4 GB, 300-cycle latency
Network	8×8 2D Mesh XY Routing

synthetic traffic patterns, on 8x8 2D mesh topologies with XY dimension-ordered routing. The injected traffic consisted of synthetic traffic that covered two types of packets to mimic realistic system scenarios: 1-flit short packets (just like request packets in a CMP), and longer 5-flit packets (just like response packets carrying a cache line). We experimented with the following synthetic traffic patterns [31]:

- Uniform Random (UR): all network nodes send packets to every destination node with equal probability
- Localized (LC): 75% of the overall traffic is local (i.e., the destination is one hop away from the source), while the remaining 25% of the overall traffic is uniform-randomly distributed to the non-local nodes
- Bit Complement (BC): each source node with address s only generates packets towards the destination node whose address bits are the complement of the source, i.e. $d = \not(s)$
- Transpose (TS): a source node s only generates packets to one destination d, for which: $d_i = s_{i+b/2} \mod b$, where s_i and d_i are the i-th bits of the source and destination address, respectively, while b is the width of their addresses in bitss

Customized scenarios, covering mostly hotspot traffic, where specific nodes receive significantly more traffic than the rest of the network, were also conducted in cases where a deeper insight on the special characteristics of each architectures was necessary.

To strengthen our experimental results, we have also conducted simulations of a 64-core tiled CMP running real application workloads on a Linux operating system. Our proposed architectures where compared both in terms of packet latency, and execution time. The simulation

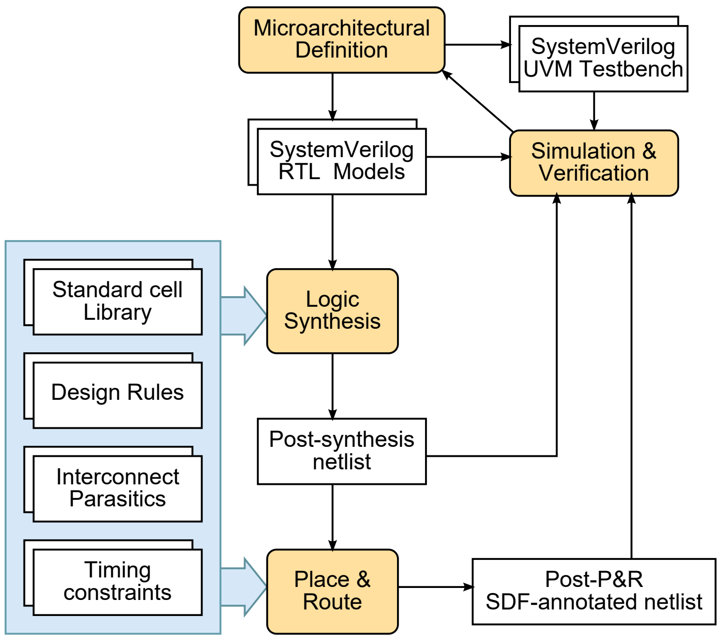


Figure 1.11: Hardware implementaton flow followed for our developed designs.

framework employs Simics – which handles the functional simulation tasks – extended with the GEMS [87] that provides a detailed timing model of the memory hierarchy and it includes the GARNET [5] cycle-accurate NoC simulator. The system configuration is shown in Table 1.1.

1.4.2 Hardware Implementations & Verification

All proposed router architectures were implemented in hardware and compared with their state-of-the-art alternatives in terms of (a) operating frequency, (b) occupied area, and (c) power/energy consumption. The implementation flow used in all cases is shown in Figure 1.11.

The RTL models of all architectures were developed in SystemVerilog HDL, and synthesized, placed and routed to a 45 nm, 0.8 V standard cell library, under worst case conditions 120 ^{o}C, using Cadence RTL Compiler & Cadence Encounter tools. The area, delay, power and energy figures were obtained for all designs, after constraining appropriately the logic-synthesis and backend tools using the same parameters for all designs under comparison.

During synthesis and placement-and-routing, we followed a mixed V_t approach, where a mix of regular V_t (RV_t), Low-V_t (LV_t), and High-V_t (HV_t) cells were used for the various implementations. LV_t cells can

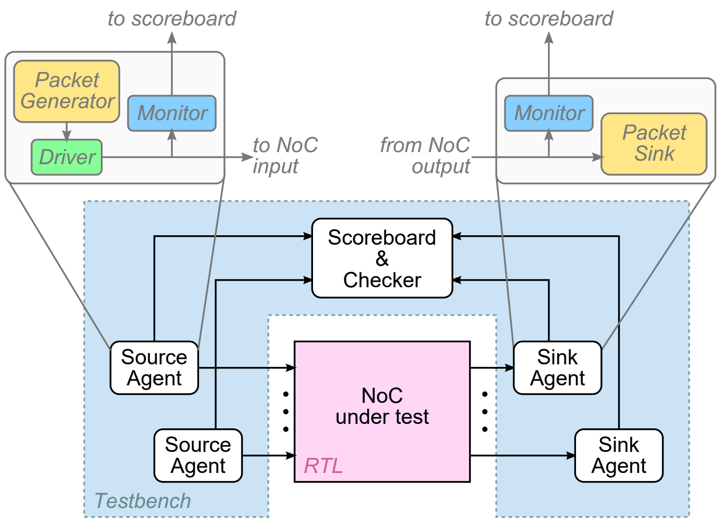


Figure 1.12: UVM-based NoC testbench organization.

switch at a much faster speed than HV_t cells, at the cost of extra leakage power. All circuits were optimized by using LV_t cells on critical paths, while HV_t cells were employed on the non-critical paths to reduce leakage power [65, 84].

For validating the functional correctness of our RTL models, we developed a UVM-like testbench structure, which was easily portable from one project to the next. Although it was developed in native SystemVerilog without the help of the UVM library [11], the testbench structure and operation relies on fundamental UVM organization shown in Figure 1.12. At the packet injection points, an agent produces random packet sequences and drives the NoC inputs. Ejected packets are sent to the scoreboard, which cross-checks them with the injected ones, reporting whether an error has occurred. Error causes may include simple ones, such as packet arrival to an unexpected destination, to more complex ones, such as packet re-ordering.

To strengthen our verifiation results, we have also employed formal methods through SystemVerilog Assertions (SVAs). Although assertion properties are not suitable for providing full proofs for complex designs, they are still a powerful tool for "bug hunting," especially during the early stages of the microarchitectural definition, or when new features are added in the design [118]. Since SVAs are attached in the RTL code in a distributed manner, bugs can be quickly located long before the testbench coverage results are produced, thus reducing the overall verification effort. Simulation and verification of the RTL models was conducted using Cadence Incisive Simulator and MentorGraphics QuestaSim.

1.5 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 briefly reviews existing state-of-the-art NoC router architectures and introduces the basic pipelined ShortPath architecture, including its baseline pipeline organization as well as the introduced fine-grained pipeline bypassing mechanism. The performance of the proposed architecture is compared against state-of-the-art using a full set of experimental results.

Chapter 5 presents the PhaseNoC explicit pipeline network operation which ensures isolation across traffic classes. Additionally, a generalized methodology that enables the creation of optimal phase schedules in any network topology is provided that together with Opportunistic Bandwidth Stealing , when, secure-grade isolation is not required, can significantly improve performance. PhaseNoC is evaluated through extensive cycle-accurate simulations that include synthetic traffic patterns, and execution-driven full-system simulations with real application workloads. Also, a detailed hardware analysis verify that PhaseNoC yields substantial cost savings, as compared to state-of-the-art VC-based architectures.

Chapter 3 discusses the physical scaling trends of tiled CMPs, identifying the inherent asymmetry between router and link delay. It then proposes the use of half-cycle link traversals for low-power link transfers, through wire engineering techniques. The proposed technique enables both link-power reduction as well as reduction of the number of buffers by reducing the credit Round-Trip Time.

Chapter 4 presents the RapidLink network with DDR link traversals, and the operation principles of our novel DDR Elastic Buffer that pipelines a flow-controlled DDR link in a plug-and-play manner. How RapidLink (an inherently dual-stream architecture) can handle a single-stream NoC configuration is also described.

Finally, a summary of our work including also its future research aspects is covered in Chapter 6.

Chapter 2

The ShortPath Router Architecture

NoCs have emerged as the dominant communication medium in multi-core setups, primarily due to their innate physical scalability, which simplifies system integration, and their high-performance characteristics stemming from the parallel handling of multiple flows traversing the network [18, 30, 8].

Envisioned as an integral component of all future many-core systems, the NoC will – inevitably – consume a non-negligible portion of the overall design's area/power budget. Consequently, it is imperative that the hardware overhead incurred by the NoC is contained to sustainable levels, while still satisfying all performance requirements. In fact, the conflicting demands for low area/power and high performance constitute the crux of many research efforts in NoC design and development.

Virtual Channels (VCs) are an integral part of most NoC architectures, since they contribute to enhancing performance, and to providing flow isolation, which is needed when providing QoS guarantees, or when partitioning the system [27]. VCs are also instrumental for the correct operation of higher-level mechanisms. For example, the cache coherence protocols of CMPs categorize all cache traffic into certain message classes (e.g., request/reply). In order to avoid protocol-level deadlocks, the coherence protocols require that the various message classes are isolated and have their own dedicated buffers [87]. This segregation is typically handled by the NoC through the use of VCs. Also, various adaptive routing algorithms require the use of VCs to avoid network-level deadlocks [38].

The VC-based NoC router, as the main building block of the on-chip network, has been the focus of significant research attention, and the target of several micro-architectural optimizations [31, 36]. The ultimate goal of this work is to design a scalable VC-based router architecture that strikes a cost-effective tradeoff between latency (in cycles) and clock period (operating frequency), while still offering a low-area/power implementation. In essence, this goal amounts to crafting an architecture that approaches – as much as possible – holistic optimality in terms of performance, cost, and scalability.

To achieve this goal, we first perform a detailed exploration of the NoC router micro-architecture landscape to identify the *principal microarchitectural features* that markedly impact the operational characteristics of a NoC router, and affect its scalability potential. Guided by the identified characteristics, we subsequently propose a new router micro-architecture, called *ShortPath*.

ShortPath, relies on an array of novel micro-architectural innovations, which work synergistically to optimize the router's operation. Overall, the **key attributes and contributions** of ShortPath are the following:

- A collapsible pipelined router organization, which allows for dynamic pipeline-stage bypassing, in order to incur the minimum possible latency to each traversing flit. ShortPath skips all uncontested stages and "fast-forwards" the flits to the first point of contention. As a result, the (nominally) 3-stage pipeline dynamically collapses to the minimum possible number of stages (i.e., 1- or 2-stage), based on traffic conditions. This is the first (to the best of our knowledge) architecture to guarantee that *each flit will spend the minimum possible number of cycles* in each router, based on prevailing traffic conditions under both low and high network traffic. Spending more than a single cycle in a ShortPath router happens only when flits encounter contention. ShortPath skips all uncontested stages and "fast-forwards" the flits to the first point of contention. As a result, the (nominally) 3-stage pipeline dynamically collapses to the minimum possible number of stages (i.e., 1- or 2-stage), based on traffic conditions.
- A radical re-organization of the NoC router's micro-architecture allows for minimal intra-router latency (in terms of cycles), while still enabling high-frequency pipelining. The router's allocation and switching tasks benefit from a new parallel setup, which is further aided by a pair of instrumental low-cost request queues.

- ShortPath's operation is *always productive*: even if the bypassing flit loses the arbitration at the first point of contention it encounters, it will never be forced to repeat any of the already bypassed stages. This is unlike existing bypassing approaches, which force flits to "re-wind" and traverse a longer pipe. The lack of pipeline stage repetitions is the result of the *always non-speculative* operation of ShortPath, which allows each flit to keep (and not kill, as done in speculative architectures) any of the already allocated resources, even if the allocation of the subsequent resources – deeper in the pipeline – is delayed due to contention. This behavior ensures consistent effectiveness (without the performance variability instilled by speculation) under *all* traffic conditions.

The experimental results show that ShortPath outperforms Scorpio [34] – the most efficient pipelined router organization with pipeline-stage-bypassing capabilities – by up to 30% in terms of latency and 25% in terms of throughput, with *no* additional hardware cost (i.e., no adverse impact on area/power/delay). In fact, ShortPath's new micro-architecture yields a shorter critical path than Scorpio [34], which translates to an equivalent increase (if desired) in the maximum operating frequency.

2.1 The Landscape of VC-based NoC Routers

The design space of VC-based NoC routers has evolved significantly over the past decade. Our goal in this section is to summarize the state-of-the-art router micro-architectures and to isolate those design features that promise the highest level of scalability, both in terms of network performance (latency/throughput), as well as hardware complexity (delay/area/power). These salient attributes will subsequently shape the proposed ShortPath router architecture.

Despite a multitude of proposed NoC implementations and incarnations, the functionality expected from a VC-based router's microarchitecture remains the same: coordinate and direct the flow of packets from the inputs to the outputs of the router. An abstract illustration of the generic VC-based router architecture is illustrated in Figure 2.1.

Arriving packets are written into the input VC buffers of the router (Buffer Write – BW). In the following cycles, they must find their way to the proper output port, after going through several allocation steps [36]. The head flit of a packet first calculates its output port through Routing

Computation (RC). Note that the selected output port could have been pre-computed in the upstream router, using Look-ahead RC (LRC) [44]. Once the desired output port is known, the head flit uses it to allocate an output VC (i.e., an input VC in the downstream router) in VC Allocation (VA). All flits of the packet try to gain exclusive access to a router's output port, on a cycle-by-cycle basis, through Switch Arbitration (SA). The winners of the SA stage traverse the crossbar during Switch Traversal (ST), and are then written in an output pipeline register. Finally, the flits move to the next (downstream) routers through Link Traversal (LT).

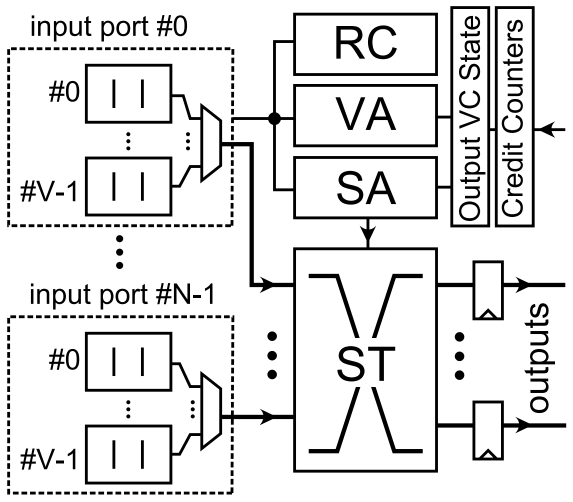


Figure 2.1: The main units of a generic VC-based router.

In this paper, we consider that the LT stage occurs in a separate cycle from the remaining operations of the router. In cases where the delay of the links is low, LT can be performed in the same cycle as ST, or even together with the allocation steps. Although this approach may be preferable for tiled systems with regular layout [131, 34] and short wire lengths, it is not the safest option – in terms of timing closure – in the general case of complex SoCs [112]. In those cases, the layout of the system is not regular, and the length of certain wires can assume arbitrary values. The wires may be pipelined to achieve the required clock frequency, or they may even connect routers that belong to different clock domains [102]. Late changes to the physical placement of the modules of the chip may completely change the critical timing paths. The NoC should be flexible enough to adapt quickly to any design changes; in extreme cases, even the topology of the NoC could change. Thus, when maximum flexibility is needed, keeping the timing of the LT stage isolated from the internal timing paths of the router is the safest approach to follow. This approach is more pronounced in low-latency NoCs built with high-radix routers, which connect modules placed far apart from each other, using long wires [68].

2.1.1 Baseline allocation

The router can implement all the required allocation steps in one cycle, or in multiple cycles, when following a pipelined organization. In either case, the designer's goal is to identify the dependencies across the allocation steps, and to minimize them by allowing the required tasks to execute in parallel, as much as possible. The parallel execution of the allocation is not always straight-forward and involves tradeoffs between the achieved throughput and the delay of the corresponding circuits that implement the router.

In the baseline case, the VA and SA stages are executed serially in the same cycle, as shown in Figure 2.2(a). The head flit of a packet selects one output VC to be requested – among possibly many candidates – in VA1. Then, each output VC selects at most one input VC requesting it, in VA2. Upon completion of VA, the SA stage commences. During SA1, every input port independently selects one input VC that will *attempt* to reach the selected output port. Thereafter, in SA2 each output port selects which input can access it. The winners of both the SA1 and SA2 steps traverse the crossbar in the ST stage, and are written to the output register. The serial execution of the arbitration steps limits the operating frequency of the router, with VA2 being the slowest step. More speed can be gained by circuit-level optimizations, such as the one presented in [37], which merges SA2 with the multiplexing of ST.

2.1.2 Routers with speculative allocation

Speculative allocation [101, 93, 16] speeds up the allocation process by speculatively performing SA, in parallel to VA2, thereby allowing a flit to arbitrate for port access, even though it has not been allocated an output VC yet. In the case of mis-speculation, a packet (still not owning an output VC) wins in SA, but loses in VA2, and the output port is left unused in the current cycle.

In order to decrease the probability of mis-speculation, the non-speculative requests (i.e., those made by packets that already own an output VC) have a higher priority over speculative ones. However,

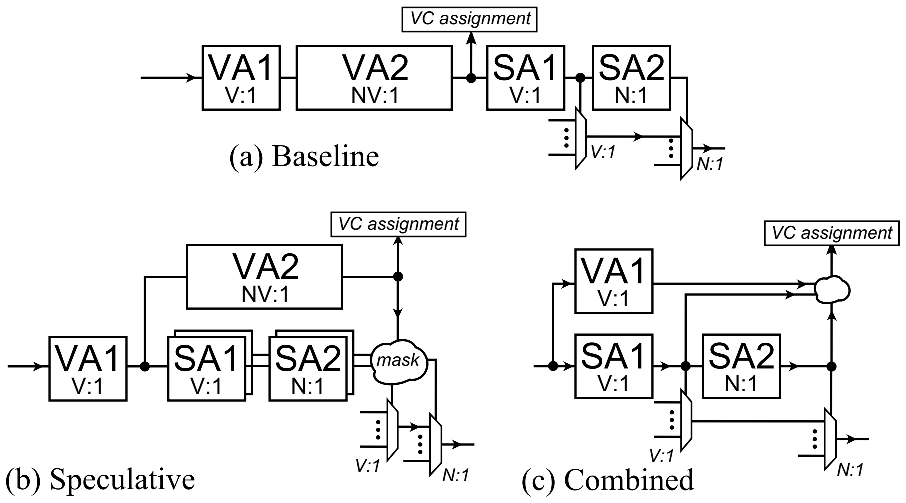


Figure 2.2: The order of arbitrations and their dependencies in: (a) Baseline, (b) Speculative and (c) Combined allocators.

achieving this requires two full SA units that separately serve speculative and non-speculative requests, and extra logic that handles the two allocators' conflicts. The VA1 stage is still done in series with SA, since we need to know which output VC is selected by each input VC, in order to check its availability (needed in VA2) and the existence of the required downstream credits (needed in SA1 and SA2).

Speculative allocation decreases the router's critical path relative to the baseline, as illustrated in Figure 2.2(b), and increases its operating frequency without incurring any performance overhead.

2.1.3 Oblivious arbitration

Credit – or output VC – availability checking can be performed in parallel to the arbitration steps; we call this action *oblivious arbitration*, and it entails hidden speculation. Under oblivious arbitration, the arbiters may grant an input request that should have been masked beforehand, since it refers to either an unavailable output VC, or an output VC without available credits. At very low input traffic, oblivious arbitration does not cause any problems, since, most of the time, the output VCs are available and their buffers are always not full, i.e., credits almost certainly exist. However, at high loads, this may create significant number of idle cycles, since many grants are killed after arbitration, because they refer to unavailable resources. If the requests to unavailable resources are masked beforehand, then this problem disappears.

The impact of oblivious arbitration is expected to be more pronounced in the SA stage, because, in this case, the output may be left unused, as a result of not checking beforehand the existence of credits. In VA, oblivious arbitration is not expected to degrade performance that much, since a failed arbitration results in a failed input-VC-to-output-VC matching; the output port may still be used by another packet, which is assigned to another output VC.

Even if oblivious arbitration may introduce un-necessary idle cycles, it offers significant benefits, in terms of the router's clock frequency, irrespective of the structure of the allocator. The process of checking for credit availability requires checking the credit counters of *all* output VCs (i.e., V per output port $\times$ N output ports) and selecting the one that corresponds to the flit's destined output port and assigned output VC; this is an N$\times$V multiplexing process. Oblivious arbitration hides the delay of this multiplexing by executing it in parallel to the arbiters of VA or SA.

2.1.4 Combined allocation

Combined allocation [75, 16, 82, 95] moves one step forward – beyond speculative allocation – by removing the VA2 stage completely. It simplifies the VA process by combining its functionality with that of SA. This simplification is only possible by restricting how many input VCs are allowed to get assigned to an output VC in the same cycle; combined allocation allows at most one VC per input to allocate an output VC in each cycle. Output VC assignment now happens in the SA stage, where flits may participate, even if they do not own an output VC yet (e.g., newly arrived head flits).

A head flit selects an output VC to request in VA1, and it is allowed to claim it in SA, as long as it is available. If it wins in SA, it receives access to the selected output port and, concurrently, it is assigned to the selected output VC. As elaborated in [117], VA1 can be performed in parallel to SA1, by exploiting the fact that it is not necessary to know that the selected output VC is available *and* with credits, but it suffices to know that at least one of the candidate output VCs fulfills the necessary conditions. The resulting path is shown in Figure 2.2(c). If a flit loses in SA2, the output is not left unused, since another flit from another input can use the selected output. In some implementations based on combined allocation [75, 74], the VA1 stage is also removed completely, and the allocated VC is received by a queue/pool of free VCs at the output. In such cases, an input VC is not allowed to make a specific output VC selection, which may be required when VCs form VNs.

Routers employing combined allocation are the simplest to implement, while they reduce the router's critical path and its area to the minimum, especially when combined with shared buffering architectures [116, 17]. However, the requirement that at most one VC per input is able to allocate an output VC – since the assignment is done concurrently with the assignment of an output port during SA2 – leads to worse matching efficiency than baseline routers. In baseline routers, many input VCs get assigned to their corresponding output VCs in the same cycle, even if only a subset of them will reach their destined output ports. Therefore, this pre-assignment of output VCs may, at first glance, seem redundant. However, the more input VCs are already assigned to an output VC, the more requests can be made in the SA stage. Therefore, the SA stage has more options in achieving an almost full input-to-output port matching, leaving fewer idle output ports per cycle.

2.1.5 Pipelined router organization

In order to meet tighter timing constraints, the NoC routers may be pipelined to increase their clock frequency [36]. Moreover, pipelining is also needed in energy-constrained cases, whereby the NoC should be able to sustain an acceptable operating frequency, even under lowered voltage. Scaling the voltage of the circuit is a useful technique for reducing power consumption. However, lowering the circuits' voltage increases the delay of their constituent logic blocks, which limits the maximum clock frequency of their operation. Pipelining retrieves some of the lost MHz in clock frequency (due to voltage scaling), thus keeping a balance between energy efficiency and achievable performance.

The baseline router can be pipelined by executing the VA, SA, and ST operations in separate pipeline stages. Although pipelining markedly decreases the delay, it also increases the packets' zero-load latency. The credit Round-Trip Time (RTT) is also increased, which results in increased buffering requirements, and, consequently, an increase in the total area of the network. Additionally, every pipelining decision stops across the borders of the traditional basic blocks within each router, i.e., VA, SA, and ST. The fact that such blocks do not have an evenly balanced delay profile makes pipelining even harder, since the achieved clock frequency is limited by the delay of the critical path. For example, the critical path tends to be dominated by the VA stage, which does not justify splitting the SA and ST into different pipeline stages, or pipelining routers with speculative allocation.

Routers with combined allocation offer finer-grained pipelining, by splitting the SA1 and SA2 phases into different pipeline stages [98, 12, 74]. The ST stage can either occur in the same cycle as SA2, or in a different pipeline stage. Combined allocation is inherently non-speculative when the SA1 and SA2 steps are executed in the same cycle. However, when SA1 and SA2 are put in different pipeline stages, a new form of speculation appears.

For example, a head flit that needs to reach output VC#0 – which is currently available – performs SA1 and wins in cycle 0. In the next cycle, i.e., in the next pipeline stage, the flit participates in SA2, but loses. If the flit only retries the SA2 step (assuming that it remains the winning flit in SA1), then output VC#0 may no longer be available. Thus, the flit is unable to move forward and should be killed, until output VC#0 becomes available again [12]. In this case, the SA1 winning step is not used productively (behaving like mis-speculation), since a flit is allowed to compete for access to an output port in SA2, without having reserved exclusive access to an output VC. Equivalently, the same situation may arise again, if we send the flit back to SA1 and let it retry both the SA1 and SA2 stages in the following cycles.

2.1.6 Pipeline bypassing

Pipelined routers employing combined allocation push the router's operating frequency to the limits, since the slowest pipeline stage consists of merely a single $V : 1$ (in SA1) or $N : 1$ (in SA2) arbitration process. However, multiple cycles (2 or 3 in the fastest cases) are needed for a flit to traverse a router's pipeline. Bypassing pipeline stages is a common practice for reducing the number of cycles spent within each router, when the traffic reaching the router is low. For instance, the 3-stage pipelined router presented in [12] and elaborated in [13] executes SA1, SA2, and ST in different stages, and allows flits to bypass SA1 when only one input VC is active within a specific input port.

SWIFT [74] and Scorpio [34] enhance the bypass option by taking advantage of the fact that SA2 and ST are executed in different pipeline stages, i.e., SA2 finishes one cycle before ST. Hence, the result of SA2 is augmented with additional header information, bundled in so called *Look-Ahead (LA) signals*, and sent to the downstream router to set up the allocation decisions one cycle prior to the arrival of the actual flit. In this way, the incoming flit is able to bypass all intermediate arbitration stages and directly perform ST in the next router. However, if this lookahead procedure fails for any reason (mostly due to contention from other inputs), the flit has to *traverse all 3 stages of the pipeline*.

In cases where multiple LA requests are contending for the same output port, the winner is declared through a simple masking process, named LA Conflict Check (LA-CC), instead of a full-fledged arbitration, in an effort to reduce the bypassing delay overhead as much as possible. This approach however, leads to *unproductive decisions*, where none of the contending flits manages to actually succeed in bypass, but instead, all of them are forced to "re-start," in order to contend again in the normal pipeline 3 cycles later.

Additionally, even in normal (non-bypass-capable) pipelined operation, not all allocation steps are used productively when the SA1 and SA2 stages are performed in different pipeline stages. For example, assume that a flit that needs to reach an output VC – which has currently enough credits available – performs SA1 and wins. In the next pipeline stage, the flit participates in SA2, but loses. If the flit only retries the SA2 step (assuming that it remains the winning flit in SA1), then the selected output VC may no longer have credits available. Thus, the flit is unable to move forward and its request – that is now obsolete – should be killed, until the credits of the corresponding output VC are increased [74, 34, 12]. Thus, the SA1 winning step is not used productively (behaving like mis-speculation), and the allocation process should restart from the beginning.

Scorpio [34] constitutes the latest rendition of the three key microarchitectural principles, i.e., combined allocation, pipeline bypassing, and look-ahead signaling, and removes many of the deficiencies of its predecessor [74]. For example, Scorpio uses the same bypass mechanism and LA signals as SWIFT, but in a more cost-effective way, without resorting to Token Flow Control [76], as done in SWIFT [74], which increases buffering requirements and incurs significant wiring overhead across routers. More importantly, Scorpio's critical path is improved, thus reaching higher clock frequencies, while still retaining the favorable characteristics of a balanced pipelined organization and efficient pipeline bypassing. However, Scorpio remains a *speculative* architecture and still suffers from the *inefficiencies of LA-based bypassing*.

2.1.7 Summary

Ideally, the design of a highly efficient NoC router architecture should combine – in a balanced manner – the benefits of all state-of-the-art router organizations. For example, the router's operation should be non speculative by construction, both when applied to single-cycle routers, and to any pipelined organization (**Design Principle 1**). It is desirable to provide similar levels of performance – and at similar cost – to speculative architectures, but without the performance variability instilled by speculation. Achieving such behavior is non-trivial and requires a careful redesign of the router's operation, especially in the case of pipelined routers that execute the SA1 and SA2 steps in different pipeline stages.

At the same time, in an efficient router organization, it is preferable to follow a structure similar to that of combined allocation, in order to ensure a very fast implementation (even in single-cycle form) and low cost, in terms of area and power (**Design Principle 2**). The combined allocation approach would ensure that both the router's area and its delay are proportional to either the number of input ports (N), or the number of VCs per input (V), and *not* their *product* (as in routers with baseline allocation, or in speculative designs that employ a VA2 stage of arbitration).

In the case of pipelined routers, the pipeline stages should be bypassable in a fine-grained manner, without the need for lookahead signals (**Design Principle 3**). The latter (a) do not scale well with high radices and long inter-router wire lengths, and (b) their overhead cannot be amortized when using small-width data channels, In the ideal case, a flit traversing routers with bypassable pipelines should not spend more cycles in the NoC than a flit traversing single-cycle routers and stalled only due to contention.

The following section demonstrates how the proposed ShortPath architecture embraces and cost-effectively implements all three salient design principles.

2.2 The Proposed Router Pipeline

This section describes the basic ShortPath pipelined micro-architecture, which comprises three pipeline stages, as depicted in Figure 2.3. ShortPath is – by construction – *non-speculative* by imposing to head flits a certain order of acquiring the needed resources. A head flit should first allocate an output VC for the whole packet, and then gain access to its output port.

VC allocation is performed in the first pipeline stage (only applicable to head flits) that executes the VA1 and VA2 stages in parallel, driven by a newly introduced FIFO structure, called the *VA Request Queue (VARQ)*. Whenever a head flit arrives at an input port, necessary *control* information is enqueued in the corresponding VARQ. The exact operation of VC allocation is analyzed in Section 2.2.1, while the details of the VARQ's operation are presented in Section 2.2.2.

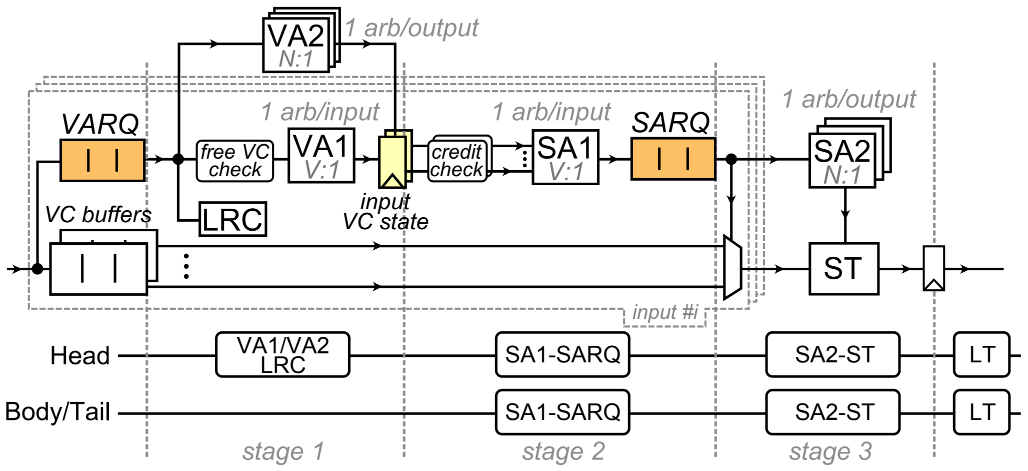


Figure 2.3: The organization of the three-stage pipelined ShortPath router, showing both the input and output arbitration logic, which guarantees non-speculative operation at low cost, and the operations performed for each flit in each pipeline stage.

The second stage allows all flits to perform SA1 and push their requests into the novel *SA Request Queue (SARQ)*, which separates the SA1 and SA2 arbitration stages. Requests to the SA1 stage can only be made by flits that already own an output VC and refer to an output VC that has available credits. Pushing the requests that win in SA1 in the SARQ makes the allocation decisions always productive. Even if a request is not satisfied in SA2, due to contention, it does not have to replay the previous pipeline stage, and it stays in the SARQ. In the third pipeline stage, all flits perform SA2 and the ones that win traverse the crossbar (ST), and are written in the output pipeline register that separates router operation from link traversal (LT). If desired, the SA2 and ST modules could be merged into one combined fast module, using the structure of [37]. The details of the switch allocation performed in the second and the third pipeline stages of ShortPath are clarified in Section 2.2.3.

Although the general order of resource allocation resembles that of a baseline router architecture, ShortPath's allocation logic (1) is significantly simplified, (2) allows for fine-grained pipeline partitioning of the allocation tasks, and (3) enables an always-productive pipeline-stage bypassing mechanism. Note that three-stage router traversal is the *maximum* (worst-case) pipeline depth that a flit may experience. As will be described in Section 2.3, ShortPath's pipeline is *dynamically collapsible* to the minimum possible depth allowed by the prevailing traffic conditions. Thus, flits may traverse a single-, two-, or three-stage pipeline, based on the encountered contention.

2.2.1 Virtual-Channel Allocation

In ShortPath, acquiring an output VC (relevant only for the head flits of each packet) is performed in the first pipeline stage and is governed by two allocation rules that greatly simplify the hardware implementation of VC allocation, without affecting negatively networking performance: in each cycle, (a) at most one VC per input port is allowed to allocate an output VC, and (b) at most one output VC may be allocated per output port.

Based on the first rule, only one input VC (still not owning an output VC) from each input port is allowed to allocate an output VC. Specifically, the packet chosen to participate in VC allocation is the earliest arriving one. The order of arrival is maintained through the VARQ, as shown in Figure 2.3, which enqueues the necessary control information for every arriving head flit. The VARQ only stores **a few bits of control information** for each head flit, i.e., its VC ID, its destined output port (as calculated in the previous router during LRC), and 2-4 bits of metadata required for the VARQ's operation. The actual flit still resides in its designated VC buffer. Once the packet succeeds in allocating an output VC, the control information is dequeued from the VARQ and the next head flit (in order of arrival, as preserved by the VARQ) can allocate an output VC to its destined output port.

Since at most one input VC is allowed to try to allocate an output VC from each input port, and only one output VC can be allocated per output port in a single cycle, there is no need to make any distinction between requests made for different VCs of the same output port. It is merely necessary to select the input port that will receive an output VC in the specific output port. Thus, one arbiter per output port suffices to handle all the requests for that port, even if they refer to different output

VCs. This results in a VA2 arbiter (one per output port) with only N inputs (requests), where N is the number of input/output ports in the NoC router. Thus, as opposed to a baseline organization, ShortPath reduces both the number of VA2 arbiters per output (one versus V) and the requests that each arbiter receives (N versus $N \times V$). Even if the baseline VC allocation allows for a marginal increase in throughput, its delay complexity and the delay imbalance that it causes across the pipeline stages, do not justify its adoption.

Once the head flit of an input VC (at most one per input port) wins in VA2, it is assigned to the output VC selected in parallel by VA1. If a packet is allowed to change VC while moving from router to router (i.e., in-flight), the head flit should prepare a vector of candidate output VCs from its destined output port, while the unavailable ones should be crossed off. From those VCs left, the head flit of each packet should choose one. Thus, a single $V : 1$ arbiter per input is enough to perform VA1.

2.2.2 VARQ Attributes and Operation

The VARQ is a low-cost hardware queue, which complements the VA operation and maintains the order of packet arrival into each input port. Each incoming *head* flit (i.e., a new packet) stores its VC ID and destined output port ID into an empty slot in the VARQ. The head-of-line entry of the VARQ feeds this pertinent control information to the VA logic of the router, thereby enabling the corresponding head flit to compete in the VA stage for an output VC.

Since packets from different VCs enqueue control information in a single VARQ, it is imperative to ensure that this serialization does *not* create any dependencies among the VCs. The VARQ achieves this feat by employing a *rotation mechanism*. If the packet at the head-of-line position of the VARQ is unable to find a free output VC (i.e., all output VCs are already allocated), the VARQ "rotates" this entry by sending it to the back of the VARQ. This rotation allows the next VARQ request in line to move into the head-of-line position. Thus, if a packet is unable to find available VC resources, subsequent packets belonging to different VCs will not be blocked indefinitely from participating in VA2. Since *a packet of one VC can never block the progress of a packet from another VC* (due to the rotation mechanism), the proposed VARQ *guarantees* the absence of across-VC dependencies and, thereby, ensures deadlock freedom. Moreover, the rotation mechanism preserves all the Quality-of-Service (QoS) properties potentially imposed by switch allocation, when VCs are used as an isolation mechanism for the various QoS-driven message classes in the system.

However, the rotation process allows requests (already enqueued in the VARQ) that do not correspond to head-of-line packets in the actual VC buffers to appear at the head of the VARQ. Additionally, the continuing arrival of new packets – and, thus, new VARQ entries – while the rotation takes place may mix up the request order within the VARQ. Hence, the VARQ employs a simple mechanism to keep track of the VARQ request order to ensure that no VC allocation request is generated, unless it refers to a packet that is at the head of its corresponding VC buffer.

Each VC keeps two separate counters, each one storing a so called *ticket* #: (1) the *next ticket* # counter, and (b) the *HoL ticket* # counter. The size of both counters is extremely small (a few bits wide), since the *ticket* # field is bounded by the VC buffer depth. These two counters are part of the **three basic rules** governing the operation of the VARQ:

- Whenever a new packet arrives at a VC, it enqueues the following control information in the VARQ: (a) its VC ID, (b) its destined output port, and (c) the *ticket* # value currently stored in the *next ticket* # counter of the corresponding VC. This ticket # indicates the new packet's order of arrival among packets of the same VC. Once the new incoming packet receives its ticket #, the *next ticket* # counter is incremented.
- The request at the head of the VARQ is allowed to participate in VA2 only if its *ticket* # matches the value in the *HoL ticket* # counter of the corresponding VC. In other words, the request at the HoL position of the VARQ is only allowed to compete in VA2 if it belongs to the HoL packet in the actual VC buffer. If not, the **request rotates** to the back of the VARQ.
- If the *ticket* # of the request at the head of the VARQ matches the value in the *HoL ticket* # counter, the packet is allowed to participate in VA2. If there are *no available/free output VCs*, the HoL VARQ **request rotates** to the back of the VARQ. If there is at least one available output VC, the packet competes in VA2. If the packet succeeds, the *HoL ticket* # counter is incremented (to point to the next packet eligible to perform VA). If the packet loses in VA2, it retries in the following cycle, as long as the requested output VCs are still available. Simply losing in arbitration is not a reason for rotating at the end of the VARQ.

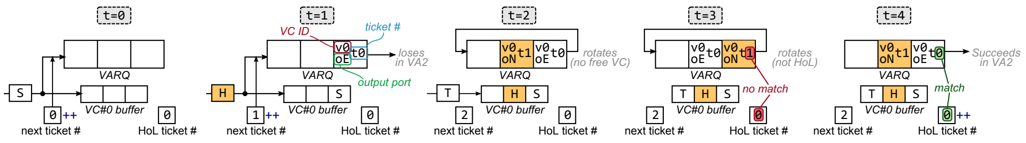


Figure 2.4: A cycle-by-cycle example of the operation of ShortPath's VARQ. Only the activity of a single input VC is depicted here for clarity.

A cycle-by-cycle example of the VARQ operation is depicted in Figure 2.4, which focuses on the activity of a single input VC for clarity. A single-flit packet ('S') arrives at VC#0 in cycle 0, and a VARQ entry is generated, which consists of its VC ID (0), its destined output port (East), and its ticket # (0). At the end of this cycle, the *next ticket #* counter is incremented by one. In the following cycle (t=1), a new head flit ('H') arrives at VC#0 and generates a VARQ entry with the new ticket # of 1. Again, the *next ticket #* counter is incremented by one. In this cycle (t=1), the single-flit packet 'S' loses in VA2 arbitration, so it retries in cycle 2. This time (t=2), all output VCs of its destined output port are occupied, causing the VARQ to perform a rotation (as per the third rule above). However, the ticket # of the next VARQ entry in line does not match the value in the *HoL ticket #* counter, causing another rotation in cycle 3. In the next cycle (t=4), the request of the head-of-line packet in VC#0 reappears at the head of the VARQ, with a matching ticket #. Assuming that an output VC has been freed, the packet participates in VA2 and succeeds, causing a dequeue in the VARQ, and an increment of the *HoL ticket #* counter.

The total number of packets (or head flits) that can be present per input port of the router also determines the depth of the VARQ. In the worst case, when the input VC buffers are full with single-flit packets, the VARQ should hold $V \times B$ pieces of control info, where B represents the buffer depth per VC. Thus, even for small values of V and B, the depth of the VARQ grows quickly.

However, the depth of the VARQ can become significantly smaller than the worst-case scenario, by taking into account one important characteristic of NoC traffic. On average, the number of packets present in each input port is equal to $V \times B/avg\_packet\_size$. In most NoC configurations, the packets that flow in the network have multiple lengths. Read requests or control packets typically have lengths of 1 to 2 flits, while reply or write-request packets can have arbitrary sizes larger than 2 [78, 63]. The packet size also depends on the width of the links of the NoC. The average packet size is the average packet length seen by the

NoC, after taking into account the distribution of each packet type in the injected traffic. For realistic scenarios, that includes both single- and multi-flit packets selecting a VARQ depth that is significantly shallower than the worst-case of $V \times B$ is enough for not compromising performance. Hence, it is a reasonable choice to limit the depth of the VARQ by controlling the number of *packets* (not flits) that can be present in the VC buffers of each input port of the router. Note that this limitation on the number of packets has **absolutely no impact on the performance of ShortPath**, as will be shown in Section 2.4.

To limit the number of packets present in each input port, we maintain a counter at each output port of the router, which counts the number of packets (head flits) that have already passed from the corresponding output port. If the maximum packet number has been reached (this number is elaborated in Section 2.4), the VC allocator stops allocating any new output VC for the particular output port. In any other case, the VC allocation operates normally. For every head flit that leaves the output, the packet counter is incremented. Once the tail flit of a packet is dequeued from the VC buffers of the downstream router, the packet counter is notified and decreased, restoring the operation of the VC allocator for that output port. In order to avoid the scenario of a single VC consuming all of the packets of an output port, thereby introducing dependencies among VCs, multiple counters (one per VC) may be added, to guarantee that at least one packet is reserved for each output VC. The negligible overhead of these counters is accurately accounted for in the hardware evaluation of Section 2.4.1.

2.2.3 Switch Allocation

The flits belonging to packets that have already been allocated with an output VC can participate in SA, which is performed in two steps. During SA1, every input port independently selects one input VC that will attempt to reach the selected output port. Thereafter, in SA2, each output port selects which input port can access it. The winners of both the SA1 and SA2 steps traverse the crossbar (ST) and are written to an output register.

State-of-the-art implementations, such as [12] and [34], execute SA1 and SA2 in different pipeline stages, following a prediction-based allocation policy. Prediction is required, since the SA1 stage is only informed of the result of SA2 in the following cycle. If the winner of SA1 acts pessimistically and predicts that it will lose in SA2, it retries again in SA1 in the next cycle by sending a second request. If the prediction turns out to be wrong, and the first request is actually granted, then another request will appear at SA2 in the next cycle, which is possibly obsolete and should be killed (if no other flit follows in the input VC buffer, or credits are no longer available). Alternatively, if the winner of SA1 acts optimistically and does not send a second request (assuming that it will win in SA2), it may waste the successful SA1 allocation step, if the prediction turns out to be wrong and the flit actually loses in SA2.

On the contrary, ShortPath *avoids any predictions*, by guaranteeing that the winner of the SA1 stage has allocated all resources required (i.e., an output VC with credits) to be eventually granted in the next stage, SA2. Once a flit wins in SA1, necessary **control information** – the VC ID and the output port request of the winning flit – are enqueued into the SARQ (again, the actual flit data remains in the input VC buffers). The port request at the head of the SARQ feeds the corresponding SA2 arbiter, and the VC ID drives the $V : 1$ per-input data multiplexer to select the appropriate input VC's data (actual flit). If the SARQ is full, SA1 in that input port is blocked. When multiple flits exist in a certain input VC, multiple requests are sent to SA1 in consecutive cycles, causing multiple requests to be pushed to the SARQ (even if the result of SA2 is not yet known for the previous requests). Eventually, once the input succeeds in SA2, a dequeue operation takes place in both the input VC buffer and the SARQ, thereby allowing the next request in line to appear. If a request is not satisfied in SA2, due to contention, it does not have to replay the previous pipeline stage, and it stays in the SARQ, thus making the SA1 decision always productive.

A cycle-by-cycle example of the operation of ShortPath's 3-stage pipeline is shown in Figure 2.5. For clarity, the activity of only one input port is depicted. In cycle 0, a head flit (H0) arrives in VC 0, it is written in the input VC buffer (BW), and its VC ID is enqueued into the VARQ. In the following cycle, H0 successfully allocates an output VC in VA, as the following flit (B0) arrives and is stored behind it in the input VC buffer. In cycle 2, H0 performs SA1 and successfully enqueues a request into the SARQ, as seen at the bottom of Figure 2.5 (recall that only request identifiers are enqueued; no data leaves the input buffer). In cycle 3, H0 wins in SA2 and traverses the crossbar, while B0 participates in SA1 and wins, thereby enqueueing its request for SA2 into the SARQ. The same procedure is followed by the tail flit (T0) in cycle 4. At this point, however, B0 loses in SA2 by a flit from a different input port (not shown), and, thus, no dequeue operation occurs in the SARQ. The flit retries and

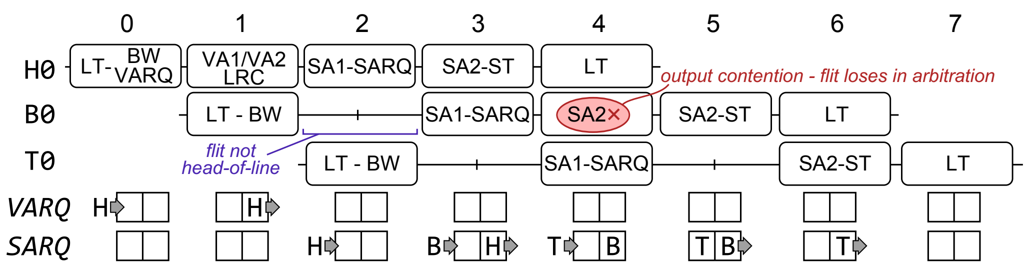


Figure 2.5: A cycle-by-cycle example of the operation of ShortPath's three-stage pipeline. The activity of only one input port is depicted here for clarity.

succeeds in cycle 5, allowing T0's request to appear at the head of the SARQ in the next cycle.

Whenever the input VC enqueues a request into the SARQ, it also consumes a credit in the selected output VC. This ensures that the input side always stays in sync with the output VC's actual buffer state. This premature credit consumption cannot create any problems, since a request in the SARQ will succeed in SA2 within a finite amount of time. This is guaranteed by updating the priority of the SA2 arbiters whenever a grant is generated; thus, once a request appears at the head of the SARQ, it is **guaranteed to succeed after at most $N-1$ cycles**. Since every request entering the SARQ (a) refers to an already allocated output VC, and (b) it has already consumed an existing credit (i.e., space is guaranteed in the downstream router), *no* entry in the SARQ can cause indefinite blocking of any other entry. Hence, **the SARQ creates no dependencies and is deadlock-free by construction**.

For credit management, we adopt the proxy counter approach of [16]. In this case, once a head flit is allocated an output VC, its credit value is copied locally (per input VC) to a proxy counter. Thus, credit checking is done locally per input VC, without additional propagation and multiplexing overhead. Multiplexing is only used for updating the value of the proxy credit counter. Note that, due to the VA policy restrictions, at most one output VC credit value from each output may be copied to the input VC side in each cycle.

2.3 Fine-Grained Pipeline Bypassing

Pipelining translates into additional cycles to a packet's latency. Even if this extra latency is amortized by the increase in clock frequency (due to pipelining), the ability to skip pipeline stages whenever possible –

with only a marginal impact to the clock frequency – will always be beneficial.

Ideally, pipeline bypassing should be enabled, or disabled, in any arbitration stage, depending on the contention encountered in each stage. For example, whenever a head flit is (a) the only flit in a certain input port, and (b) the only flit requesting a particular output port, then said flit should spend at most one cycle in the router. ShortPath's fine-grained pipeline bypassing mechanism facilitates the bypassing of *each and every pipeline stage* (whenever possible), under both low and high traffic load. In fact, any flit traversing a ShortPath router is guaranteed to spend *the minimum possible number* of cycles in each router, depending on the usage of the input/output ports.

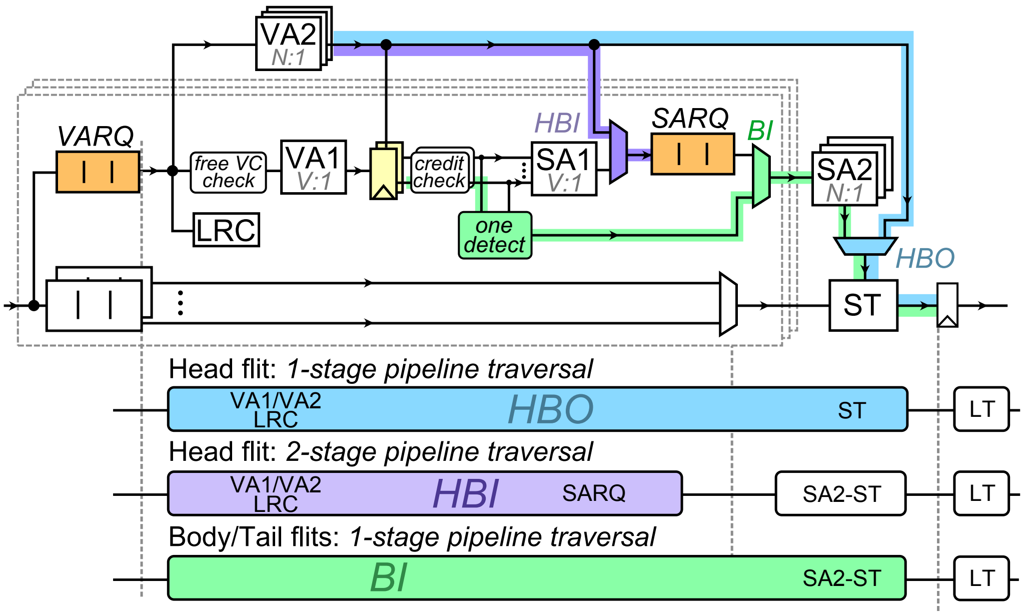


Figure 2.6: The fine-grained pipeline-stage bypassing mechanism employed by ShortPath. Each individual pipeline stage may be skipped on-demand, and any flit traversing a ShortPath router is guaranteed to spend the minimum possible number of cycles in each router, based on the usage of the input/output ports.

In ShortPath, the head flit completes VA2 and is assigned to an output VC of a certain output port. If the SA2 stage corresponding to the same output port does not see any other request, then the grant of VA2 is, actually, the only grant given by this output port to any input VC. Therefore, the *VA2 grant can be re-used* and act as an SA2 grant, too. This feature is *unique* to ShortPath, which imposes the rule that at most one output VC can be assigned per cycle. In this way, VA2 bypasses both SA1 and SA2, and it allows the head flit to move directly to ST using the Head Bypass Output (HBO) multiplexer, as illustrated in Figure 2.6 (**blue path**). Note that the HBO path refers to control signals only – the actual flit still uses the input's multiplexer to access the crossbar in ST. Additionally, in terms of hardware delay, both the normal SA2-ST path and the bypass VA2-ST path exhibit the same delay, since the VA2 step in ShortPath is implemented using just a single $N:1$ arbiter per output port (similar to SA2).

The head flit may be impeded from traversing the router in a single cycle in the following three cases: (a) the flit's destined output port is currently being requested by another flit in SA2, either from the same, or from a different input port; (b) the head flit loses in VA2; or, (c) another flit makes a request in SA1 from the same input port. In all three cases, the bypass opportunity is only lost due to contention, and not because of any micro-architectural limitation. Actually, the same behavior is expected under the same traffic conditions in a single-cycle (non-pipelined) router.

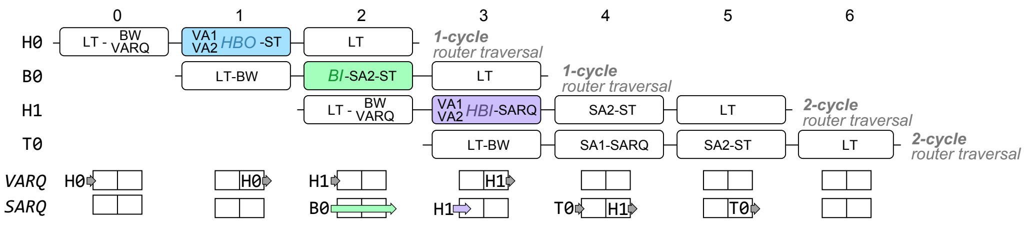


Figure 2.7: A cycle-by-cycle example of the operation of ShortPath's fine-grained pipeline bypassing mechanism. All bypass paths are exercised here, leading to single-cycle router traversal for the H0 and B0 flits, and two-cycle router traversal for H1. Due to contention, the T0 flit cannot exploit any bypassing opportunities.

It should be noted that the pipeline-bypass operation in ShortPath always makes a productive step. If, for instance, the head flit is the only one in its input port making a request, while there are many contenders for the selected output port (i.e., it cannot bypass SA2), then the flit can alternatively bypass the SA1 stage (which is uncontested) and access directly the SARQ. In that case, the head flit will be able to traverse the router in two cycles. In the first cycle, it completes output VC allocation (VA1/VA2) and moves directly in the SARQ, from where it can reach SA2 in the next cycle. This second bypass path is enabled by the Head Bypass Input (HBI) multiplexer in Figure 2.6 (**purple path**). The addition of this input bypass functionality enables the overlapped (in time) execution of VA and SA in a fine-grained manner; parallelism is even enabled for the first time across the smaller arbitration stages of VA1 ($V\!:\!1$ arbiter), VA2 ($N\!:\!1$ arbiter), and SA1 ($V\!:\!1$ arbiter).

ShortPath's fine-grained bypassing skips all uncontested stages, and "fast-forwards" a flit to the first point of contention. Even if the bypassing flit loses the arbitration at the first point of contention it encounters, it will never be forced to repeat any of the already bypassed stages. This is unlike existing bypassing approaches, which force flits to "rewind" and traverse a longer pipe [34, 74, 12]. The truly non-speculative operation in each stage of ShortPath's pipeline and the careful use of queues (VARQ and SARQ) – instead of mere pipeline registers – is the enabler for this behavior. Most importantly, this bypass functionality is achieved without changing the flow control, and without introducing any look-ahead signals across routers, as needed in [74, 34]. Additionally, whether flits follow a bypass path, or not, depends solely on the traffic conditions (absence of contention, or not) of the current cycle. There is no reliance on past-cycle (probably stale) traffic conditions, or input-output matchings, to make future bypass [34], or allocation decisions [89, 88, 24].

Spending a single cycle within the ShortPath pipeline is also possible for all flits that have already been allocated with an output VC. If a flit finds no contention at its input port, i.e., no other input VCs are active and the SARQ is empty, the flit is allowed to bypass the SA1 stage and directly participate in SA2 through the Bypass Input (BI) multiplexer, as shown in Figure 2.6 (**green path**). If no contention is encountered at the output port, the flit is granted access and departs immediately. If a flit bypassing SA1 encounters contention in SA2, it pushes its request in the SARQ, thus having the chance to retry SA2 directly in the following (or a subsequent) cycle. On the other hand, if the SA1-bypassing flit actually wins the SA2 arbitration, then no request is pushed into the SARQ.

Figure 2.7 presents an example of the operation of ShortPath's pipeline, which includes bypassing functionality. For clarity, we assume arrival of flits at only one input port of the router. In cycle 0, a head flit arrives at input VC 0, and its VC ID is enqueued into the VARQ. The flit succeeds in VA in cycle 1 and, since there is no other input VC active in the same input port, it tries to reuse the result of VA2 to immediately reach ST. Assuming there is no output contention, i.e., no other SA2 requests exist for that output port, the flit successfully bypasses SA2 using the HBO multiplexer (see Figure 2.6) and moves directly to the output. In the next cycle (cycle 2), a newly arrived body flit (B0) is the only active flit in its input port, and can, therefore, completely bypass the SA1 step using the BI multiplexer (Figure 2.6) and participate in SA2. As the flit wins in SA2 and moves forward, a new head flit (H1) arrives at input VC 1. The H1 flit succeeds in VA in cycle 3, but we now assume that its destined output port is also being requested by a flit from a different input port, thus blocking H1 from directly reaching the output. However, H1 is alone in its input port and can, therefore, bypass the SA1 stage and directly reach the SARQ through the HBI multiplexer in Figure 2.6; i.e., H1 effectively "fast-forwards" to the first contested junction in the pipeline. This scenario does not apply to subsequent flit T0, which has no bypassing possibilities in cycle 4, since flit H1 is trying to access ST. In cycle 6, all flits have left the router in the least possible number of cycles, by exploiting all bypass opportunities. The extra cycle spent by H1 is simply the result of *contention* and not a limitation of the router's micro-architecture; the output port was still utilized in that cycle, albeit by a different input port.

In general, spending more than a single cycle in a ShortPath pipelined router happens only when flits encounter contention. At ultra-low loads – e.g., as observed under bursty traffic in a single input VC – a packet's head flit would perform VA2-ST in a single cycle, while the other flits (body and tail) would perform SA2-ST, uninterrupted. As traffic increases across different input ports, VA2/SA2 contention also increases, and pipeline bypass opportunities are reduced. Flits may need to wait more cycles to be assigned an output VC in VA2, or wait for the SA1 winners in their input port to eventually win arbitration in SA2. Nevertheless, this is a latency overhead that packets would suffer in any NoC router architecture.

2.4 Experimental Evaluation

In this section, we evaluate the performance of the ShortPath architecture and compare it with Scorpio [34], which represents the most efficient router micro-architecture available in the open literature, in terms of hardware complexity and network performance.

2.4.1 Hardware complexity

For the hardware complexity evaluation, both ShortPath and Scorpio [34] routers were implemented with look-ahead RC [44], a 64-bit flit (link) width and 5-flit deep input VC buffers. For ShortPath, the VARQ was sized to six positions, meaning that it could host a maximum number of six packets per input port. Note that our network performance experiments indicate that increasing the maximum number of allowed packets to values higher than six (i.e., increasing the VARQ depth beyond six) does not yield any further improvement in latency/throughput. The SARQ can host up to two pending requests for the third pipeline stage.

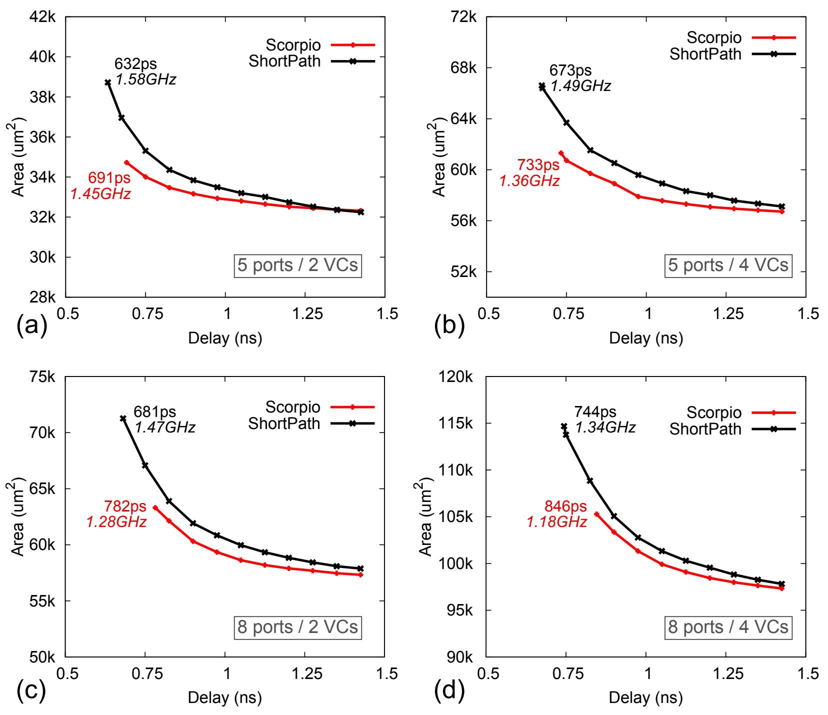


Figure 2.8: Hardware implementation (area/delay) comparison of 5- and 8-port ShortPath and Scorpio [34] three-stage pipelined organizations (including pipeline bypassing), for 2 VCs, and 4 VCs per input port.

Figure 2.8 shows the area-delay behavior of both routers under comparison, for 2 and 4 VCs per input port assuming 5-port routers, as needed by a 2D mesh network, and 8-port routers that can be used in higher-radix topologies [85], or in 2D meshes that employ concentration [15]. In all cases, ShortPath is faster than Scorpio, achieving a 8% and 12% smaller minimum delay (i.e., critical path delay), on average, for the case of 5- and 8-port routers, respectively, due to its highly parallel allocation architecture.

It should be noted that the delay numbers reported here correspond to a low voltage of 0.8 V, which significantly increases the delay of the circuits. For example, a close inspection of the clock frequency of ultra-fast, three-stage commercial routers [60, 98] optimized at the transistor level – which offers additional speed benefits over standard-cell-based design – reveals that their frequency marginally surpasses the 1 GHz mark when operated at 0.8 V.

ShortPath routers are equally area-efficient as Scorpio when sized under *equal delay*. Specifically, in the case of 5-port routers, at Scorpio's maximum operating frequency, ShortPath presents a minimal 5% area overhead, which decreases to 3% for the case of higher radix 8-port routers. Although the two architectures are based on different pipeline approaches, they eventually both require equal resources. For instance, even though ShortPath's use of the queues (VARQ and SARQ) presents an area overhead, an equivalent cost is paid by Scorpio, due to its data registers for the separate ST pipeline stage and the handling of the lookahead signals.

The hardware complexity analysis is completed by reporting the energy behavior of the routers under comparison. Energy (or area) comparisons are meaningful when the compared circuits are optimized for the same delay. Based on the delay profile reported in Figure 2.8, we select the designs that correspond to a delay of 850 ps for the cases of both 2 and 4 VCs. The energy consumed in each case is reported in Figure 2.9. The energy analysis is reported after taking into account all layout parasitics, as done in delay analysis, while the switching activity has been computed using delay-accurate simulations of the derived logic-level netlists. The evaluated routers are all driven by the same arriving packet sequence, which mimics uniform random traffic of 1-flit and 5-flit packets at an injection rate of 0.2 flits/cycle. The traffic characteristics determine the header contents of each packet, while the data contents – i.e., the payload – of each packet is produced using a uniform random number generator. In all cases, the energy required to drive the output links is also included. The results in Figure 2.9 clearly indicate that the energy efficiency of both architectures is comparable, i.e., ShortPath achieves, under equal delay, slightly larger – but comparable – energy per cycle than Scorpio, for both 2- and 4-VC configurations.

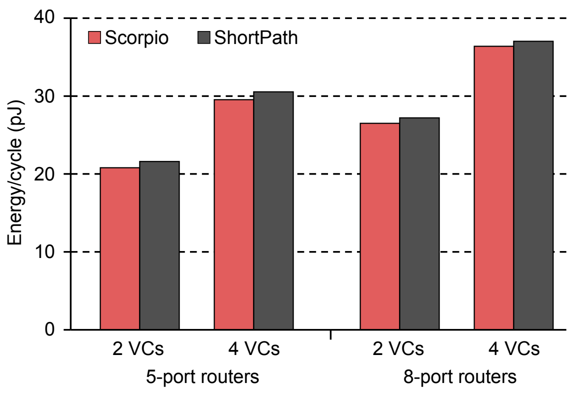


Figure 2.9: The energy per cycle (in pJ) expended in ShortPath and Scorpio [34] routers, when operated and sized under equal delay (850ps), for 5- and 8-port routers, with two and four VCs per input port.

2.4.2 Network performance

The first set of performance evaluation experiments involves Uniform Random (UR) and Bit Complement (BC) permutation traffic, according to the evaluation methodology preseted in Section 1.4. For the latency-throughput analysis, we assume a bimodal distribution of packets with 50% of the packets being short, 1-flit packets, and the rest being long, 5-flit packets. The resulting load-latency curves are depicted in Figures 2.10(a) and 2.11(a), for UR and BC traffic, respectively. In order to get a better insight into the bypassing efficiency of the two architectures and better interpret the latency results, we also plot the percentage of flits that traverse single-, two-, or three-stage router pipelines under various loads. The flit percentages for Scorpio ("SC") and ShortPath ("SP") are presented in Figures 2.10(b) and 2.11(b) for UR and BC traffic, respectively. Note that the number of pipeline stages traversed by a flit does not directly translate to an equal number of cycles spent for intra-router traversal. In fact, flits may actually spend more cycles in the router while waiting in the input buffers behind other flits, or while waiting to be granted access in various allocation stages.

At very low loads, both designs perform equally well, in terms of latency, since bypass paths are frequently activated and most flits spend a single cycle in each router. However, at medium and high loads, contention stresses the bypassing mechanism, causing Scorpio to miss bypassing opportunities, mainly due to its allocator's inherent speculation, and, partly, because of the absence of actual arbitration in conflicting LA bypassing requests. In any case, whenever bypass fails, Scorpio forces a

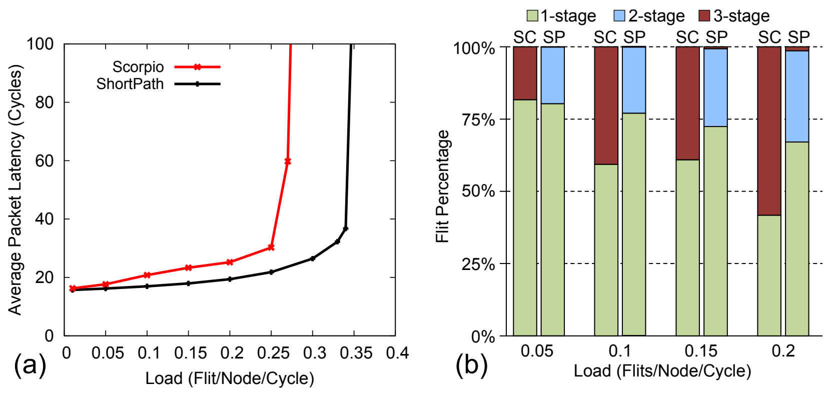


Figure 2.10: (a) Latency vs. load curves, and (b) percentage of flits traversing the 1-, 2-, or 3-stage router pipeline under various loads, for an 8×8 2D mesh network using ShortPath ("SP") and Scorpio ("SC") routers under *uniform random* traffic.

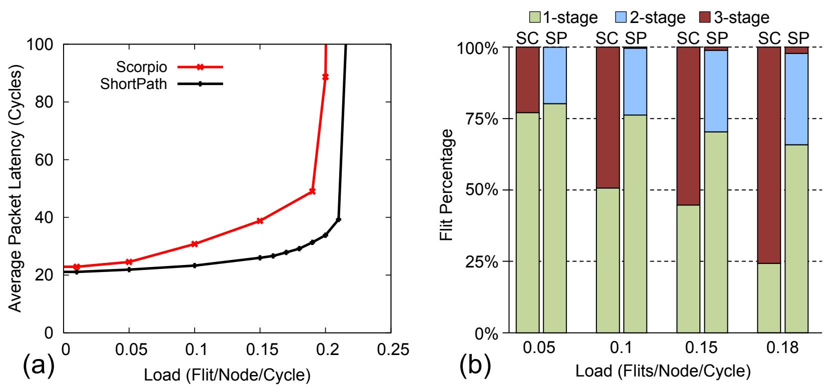


Figure 2.11: (a) Latency vs. load curves, and (b) percentage of flits following the 1-, 2-, or 3-stage router pipeline under various loads, for an 8×8 2D mesh network using ShortPath ("SP") and Scorpio ("SC") routers under *bit complement* traffic.

full 3-stage pipeline traversal, as seen in Figures 2.10(b) and 2.11(b). Instead, ShortPath dominates at such loads, exhibiting up to 30% lower packet latency at medium loads, due to its dynamically collapsible pipeline. The latter allows flits to traverse only as many pipeline stages as imposed by the first point of contention. Apart from its efficient bypass mechanism, ShortPath's completely non-speculative and always productive allocation logic allows it to sustain much higher loads, leading to increased saturation throughput by 21% and 9% under UR and BC traffic, respectively.

In addition to using purely synthetic traffic patterns, we also employ traffic patterns that are derived from real application workloads. Specifically, we employ the hot-spot traffic model from [80], which synthesizes traffic that closely resembles the traffic behavior of PARSEC application benchmarks [20] running on a CMP. Under this PARSEC-derived Hot-Spot (HS) traffic pattern, 20% of the nodes receive 50× more traffic than the rest, while the remaining injected traffic is uniformly distributed to all other destinations. Routers support 3 Virtual Networks, as required by the MOESI cache coherence protocol, with each one assigned a single VC (3 VCs in total). In order to mimic the behavior of real applications, VCs 0, 1 and 2 receive 77%, 22% and 1% of the injected traffic respectively, while packet distribution is skewed, with 1- and 5-flit packets being 70% and 30% of the total packets injected respectively [80]. As seen by the obtained results in Figure 2.12(a), ShortPath offers the highest saturation throughput, which is increased by more than 20% relative to Scorpio. It is worth mentioning that the PARSEC-derived HS traffic pattern is dominated by 1-flit packets, which significantly lower the average packet size, thereby increasing the total number of packets present in each input port, as described in Section 2.2.1. Despite this increase, however, the chosen VARQ depth of 6 was still sufficiently large to *not* generate any throttling.

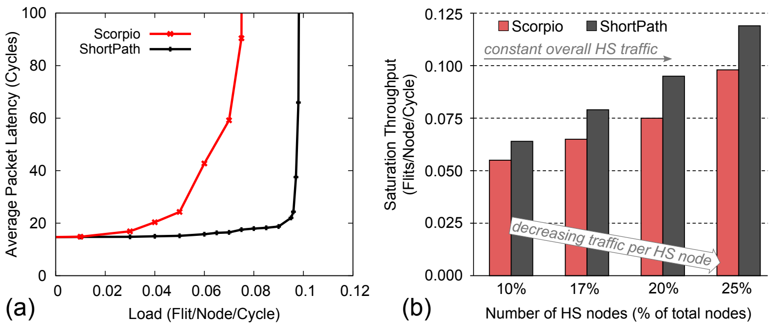


Figure 2.12: (a) Latency vs. load curves for 8×8 2D mesh networks using ShortPath and Scorpio routers, under the PARSEC-derived HotSpot (HS) traffic model of [80], which closely resembles the traffic patterns of PARSEC application benchmarks [20]. (b) The saturation throughput of both router designs when decreasing the number of HS nodes.

In order to provide a better estimation of the performance under the PARSEC-derived traffic patterns, we also test scenarios with varying number of HS nodes and measure the saturation throughput of each design. In all experiments, the percentage of HS-directed traffic, the packet distribution, and the load for each VC are held constant, as described above. Since all HS nodes always receive 50× more traffic than the rest, as the number of HS nodes is *increased*, the total traffic directed to these nodes *decreases*, thus causing less congestion around these nodes. The effect can be seen in the results of Figure 2.12(b), where the throughput of both designs increases, as the number of HS nodes increases. Still, in all cases, ShortPath outperforms Scorpio, demonstrating an average increase of 18% in saturation throughput.

It should be noted that the new micro-architecture constitutes a fundamental overhaul of the router operation, which yields improvements under all traffic conditions. Hence, a different traffic pattern (as generated by some other benchmark application) would not change the observed trends; it would simply change the amount of performance improvement achieved by ShortPath.

In summary, ShortPath outperforms Scorpio in all conducted experiments, both in terms of latency and throughput. Moreover, since ShortPath provides a faster implementation (due to its shorter critical path), if one also takes into account the **maximum operating frequency** of each design, the achieved performance gains are magnified. For example, ShortPath's saturation throughput, when **measured in Gbps**, is increased – as compared to Scorpio – by 27%, 17%, and 35% under UR, BC, and PARSEC-derived HS traffic, respectively.

2.5 Conclusions

As the number of network nodes in multi-core systems increases, NoC routers should be able to operate under high frequencies through pipelining, while, at the same time, offering low-latency packet transmission through efficient pipeline bypassing. In this work, we present ShortPath, a high-speed router architecture that manages to offer both traits: a higher operating frequency, and an efficient allocation and pipeline-bypassing mechanism that achieves singificant latency reduction and throughput increase, as compared to a current state-of-the-art router architecture [34].

These performance attributes of ShortPath stem from the novel reorganization of the router's allocation logic, which allows – for the first time, to the best of our knowledge – a truly non-speculative and parallel allocation to be applied to pipelined routers combined with fine-grained pipeline-bypass capabilities. The key distinguishing features of ShortPath (relative to state-of-the-art architectures) are the generation of always productive pipeline bypass steps, and the ability to "fast-forward" the flits to the first point of contention. These features cohesively yield a smooth linear increase in average packet latency with increasing load.

Finally, since ShortPath is orthogonal to the link-level flow control policy, and it does *not* require any look-ahead signals to achieve its bypass functionality, it can easily incorporate partitioned input speedup as a low-cost vehicle (only modest additional area, with no adverse impact on delay) to further increase the saturation throughput.

Chapter 3

Low-Power NoCs for Tiled Chip Multiprocessors

In this chapter, our work focuses on tile-based CMPs, and capitalizes on one of their key characteristics: the regularity in their physical layout. Tiling facilitates easy integration of multiple cores on the same die, and it enhances the design's scalability to increasing numbers of cores. The prevalent NoC topology in CMP designs is the 2D mesh, which is amenable to tile-based layouts [15].

Since the overall size of a CMP die tends to remain constant across different processor generations (due to yield and cost issues), any increase in the number of on-chip tiles is inevitably accompanied by a corresponding decrease in the size of each tile. This attribute has a profound impact on the system's NoC. The effective network throughput per core decreases (due to elevated cross traffic), while the average source-to-destination hop-count increases (due to increasing network diameter). Thus, both latency and throughput suffer as the NoC mesh size increases to accommodate the extra CMP tiles. At the same time, the escalating numbers of on-chip tiles inevitably affect the NoC's power budget. The router buffers and the inter-router links constitute the two major NoC power consumers [15, 58]. To sustain scalability into the many-core realm, it is imperative to curtail the power expended in NoC buffering and link traversal, without incurring any latency/throughput penalties.

On the other hand, even though the number of tiles increases, the inter-tile link distances scale *down* proportionally to the decreasing tile dimensions. Figure 3.1 illustrates this effect by comparing the link lengths of

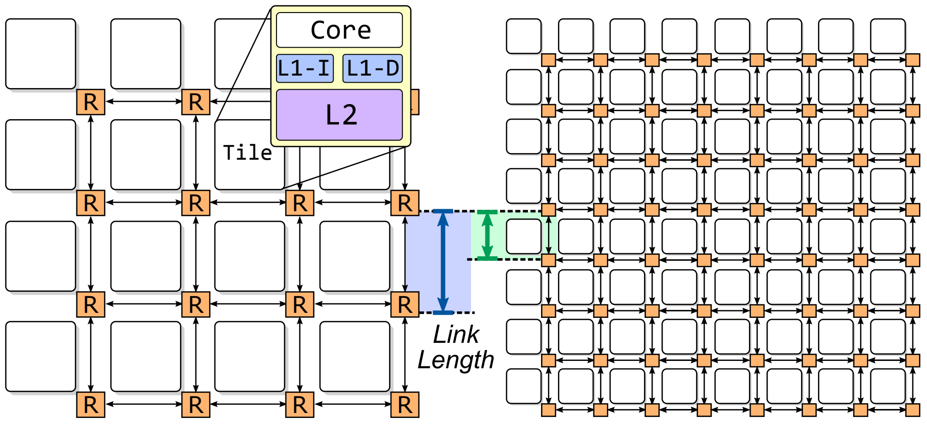


Figure 3.1: Assuming the same (or similar) die size, as the population of processing cores increases in tiled-based CMPs, the inter-tile link length decreases. This phenomenon is illustrated here by comparing a 16-core CMP (left) to a 64-core CMP (right).

two different processor generations, comprising 16 and 64 cores, respectively. For instance, at the 32 nm technology node, the longest side of a typical CMP tile (i.e., CPU core + 32 kB L1 instruction and data caches + 512 kB L2 cache slice) is 3.27 mm – as demonstrated in [120], while this distance is reduced to only 0.9 mm in 15 nm technology [1]. In systems employing simpler tiles [53], or tiles with shared L2 cache slices, the tile area will be even smaller. Thus, the inter-tile wire connections of future CMPs will hardly exceed 1 mm.

This work harnesses the swift wire traversal made possible by wire engineering to enable truly low-power NoCs for future CMPs, which are characterized by: (a) scaled inter-tile wire lengths and increasing network sizes (i.e., the tile size decreases due to technology scaling, while the on-chip tile population increases), and (b) constant, or modestly increasing, clock frequencies, due to the power-wall phenomenon. Rather than using the wire speed to cover longer distances in a given cycle [120, 1, 68, 72], we propose exploiting said swiftness to *rapidly transfer flits* between adjacent routers in *half a clock cycle*. This can be achieved by *utilizing both edges of the clock* during the sending and receiving of flits. By leveraging half-cycle link traversal we achieve a double-faceted objective:

Link-power reduction, by substantially lowering the effective wire capacitance. The latter is reduced irrespective of the data profile, and by altering only the physical layout of the link wires. The attained wire-capacitance reduction renders, in turn, the half-cycle constraint easier to satisfy.

Buffer reduction, by significantly reducing the credit Round-Trip Time (RTT), which lowers the full-throughput buffering requirements to a size smaller than the current minimum. The decrease in RTT is facilitated by a novel switch allocator micro-architecture

Both aforementioned achievements translate into tangible and extensive NoC power savings, without adversely impacting the latency/throughput performance. On the contrary, the new architecture also yields some latency improvements over a baseline NoC. The proposed design is investigated in detail and quantitatively explored to highlight its potential. Extensive cycle-accurate simulations using both synthetic traffic patterns, and execution-driven full-system simulations with real application workloads, validate the efficiency of the new low-power NoC architecture. Furthermore, detailed hardware analysis using placed-and-routed designs synthesized in a 45 nm standard-cell library corroborates the achieved significant power improvements

3.1 Motivation and Key Concepts

The delay experienced during link traversal in a NoC is determined by the capacitance and the resistance of the wires of the link, and by the manner that the wires are driven. Reducing link delay can be achieved using several approaches, such as (a) promoting NoC links to upper metal layers, (b) increasing the wire spacing, (c) using wire shielding, and (d) using across-wire repeaters.

In most systems, the NoC links can be routed only in intermediate metal layers with $2\times$ and $4\times$ lower resistance than the resistance of local routing layers. Metal layers reserved for local routing are used by the processing cores and their caches, while top metal layers (with almost $8\times$ lower resistance) are primarily occupied by power and clock routing [120, 28]. Therefore, the NoC links are accommodated within certain intermediate layers that are neither too resistive, nor too dense, thus allowing for low-delay link traversal. As measured in [120], and used in a real prototype at 32 nm, the wire delay of this group of metal layers ranges from 60-300 ps/mm, depending on repeater placement and wire spacing.

Similar results have been shown by a variety of real prototypes. For example, IBM has shown that, with appropriate wire spacing and metal layer selection, wires can cross distances of up to 2.7 mm in 210 ps at 45 nm [45], while, at the same technology node, Intel drives a wire of 5.4 mm in 270 ps [98] with appropriate wire engineering. Similar trends have been shown in [123], while, even at older 65 nm technology, 1.5 GHz could be reached by an automated design flow for 2 mm links, at minimum inter-wire distance [109]. Recently, SMART [72, 71] was demonstrated to traverse 16 mm of wire (16 hops of 1 mm each) at 1 GHz, by utilizing 1-mm-spaced repeaters and $3\times$ larger wire spacing than the minimum allowed. This translates to crossing 4 mm in less than 250 ps. Similarly, NoCs designed recently with high-radix routers assume repeated wire delays of 66 ps/mm, which are used to cross long links of 5.4 mm in a single cycle [1].

On the contrary, the delay of the routers, which involves a large range of parameters, such as the router's radix N, the number of virtual channels per port V, and the flit width W, is increased relative to link traversal. After a large set of experiments[1] using a 45 nm standard-cell library at 0.8 V, we verify that the router's delay spans 650–1100 ps, assuming fast single-cycle routers employing a combined-allocation policy [103, 82]. Hence, based on the delay profile of routers and the range of wire delays, *the delay of a router is always* $2\times$ *larger than the wire delay per mm*. Equivalently, the delay of a router corresponds roughly to wire traversals of 3–6 mm, assuming a conservative wire delay in the middle of the possible wire delay range.

Previous work tried to exploit this asymmetry between the intra- and inter-router delays (link traversal) in several ways, with the main goal being the transfer of flits over a larger distance in a single cycle. Such solutions led to either high-radix networks [1, 120] with long connecting links, or to networks that allowed flits to traverse multiple network hops of shorter wires in a single clock cycle (through router bypassing) [73, 71].

Motivated by the same asymmetric delay property (between router and link traversal), we propose a different design approach, that reaps benefits in both power consumption and network latency. By deliberately creating an artificial asymmetry between the intra- and inter-router delays of tile-based CMPs, we achieve half-cycle link traversal. By leveraging this attribute, the proposed NoC architecture can markedly reduce

[1] Evaluations were performed for the following set of NoC router design parameters: N={3,5,8}, W={16,32,64}, and V={2,4,6}. For each {N,W,V} triplet, an independent delay-constrained optimization was performed.

both the link power and the buffer size, in addition to improving performance. The designs of [21, 92] also employ both edges of the clock for link traversal. However, their aim is the simplification of clock distribution, without offering any power benefits.

3.2 Facilitating Half-Cycle Links

In the proposed architecture, adjacent routers (placed at the corners of neighboring tiles, as shown in Figure 3.2) operate under opposite clock edges. The 2D mesh topology makes the connectivity look like a checkerboard pattern. Flits have a full cycle to execute all router operations, but only half a cycle to get from one router to the next. Figure 3.3 illustrates a single-flit packet traversing three network nodes. In the top timing diagram of Figure 3.3, the network – which employs the proposed architecture – consists of white and gray nodes, which operate on the positive and negative clock edges, respectively. In cycle 0, the flit is injected and written in the input buffer of router A, on the positive clock edge. The flit spends a whole cycle executing router operations and is eventually written on the next positive clock edge in the router's output register. Half a cycle later, the flit has crossed the whole link (Link Traversal, LT) and is written in the input buffer of router B, captured on the negative clock edge. A whole cycle is again spent inside the router, until the flit appears at the output link after the negative edge of cycle 2. Router C captures the flit on the positive edge of cycle 3, allowing the flit to eventually appear in the output register, and leave in cycle 4.

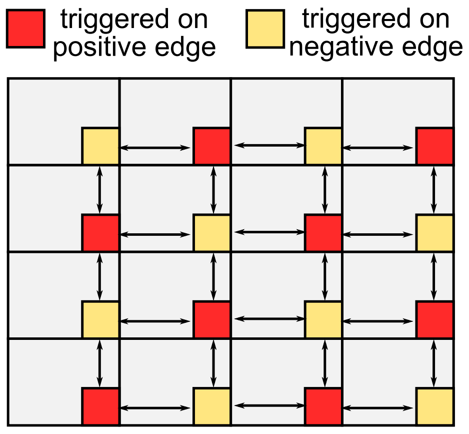


Figure 3.2: Mesh topology for a tiled CMP employing alternate clock edges for adjacent routers.

The same scenario without half-cycle links is depicted in the bottom timing diagram of Figure 3.3, in which only positive-edge routers are used. The flit spends a full cycle for both link and router traversals, eventually being ejected one cycle later than in the top diagram. Note that, in both cases, the links have the same length; scaling is merely used for illustrative purposes (to depict the timing difference)

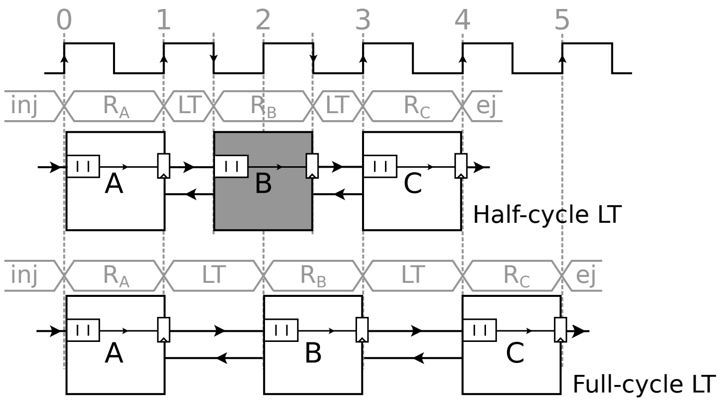


Figure 3.3: Networks-on-Chip with half-cycle links are built by restricting adjacent routers to operating on opposite clock edges. The small wire delay (for reasonably scaled inter-tile wire lengths) allows wire traversal to be completed in half a cycle.

Half-cycle links decrease the NoC's zero-load latency, using the same routers, and while working under the same clock frequency, relative to a baseline NoC with full-cycle links. In baseline NoCs with single-cycle routers (i.e., one cycle is spent in the router and one cycle on the link), zero-load latency in absolute time (number of cycles divided by the clock frequency) equals $T_B = (2H + L1)/f_{CLK}$, assuming an L-flit packet that has to traverse H hops to its destination. When the same packet traverses a NoC with half-cycle links, it saves half a cycle in each hop, resulting in a latency of $T_{HC} = (1.5H + L - 1)/f_{CLK}$. This translates to 8% to 25% latency savings, considering a packet length range of 1 to 5 flits, which travel in the NoC from 1 to 8 hops away.

Getting the most out of the new design requires a slight intervention to the clock-tree synthesis process to increase safety during timing closure. Typically, NoC clock trees are designed hierarchically [92]: intra-router clock trees first, and inter-router clock trees later. This scheme favors the paths within routers, as the clock skew of the local clock tree is minimal. However, the link traversal stage suffers from high clock skew. This is due to the high On-Chip Variation (OCV) impact of the long divergent paths to the launch and capture flops [19], as shown in Figure 3.4(a). The

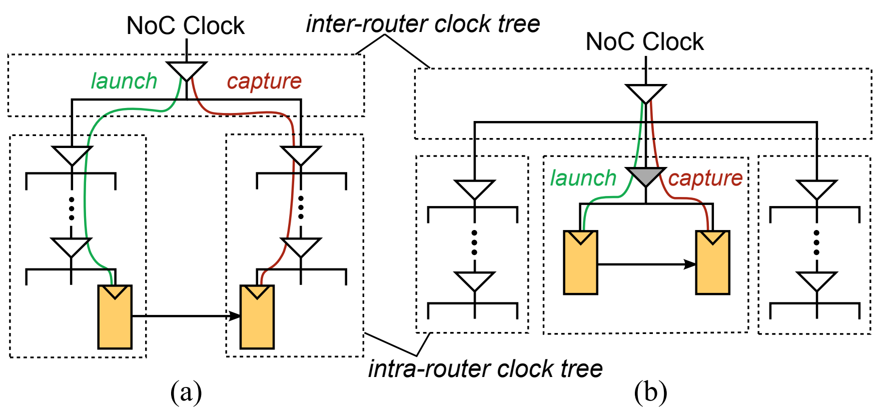


Figure 3.4: (a) A conventional clock-tree structure, and (b) the addition of a clock buffer in the middle of the link, which reduces the effect of increased timing margins due to delay variability.

timing sign-off for the link stage is more challenging, as only half the period is available. To meet this timing constraint, clock-tree synthesis should be properly guided; a clock buffer is inserted to directly drive the flops of the link, as shown in Figure 3.4(b). This enforces a third clock hierarchy, which loosens the link-timing constraints by removing the OCV-skew impact [19].

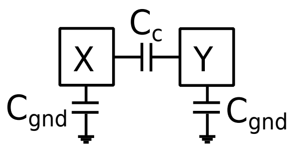
3.3 Link Capacitance Reduction

Half-cycle links allow for significant effective wire capacitance reduction without any performance overhead. Lower wire capacitance leads to lower power dissipation in the links of the NoC, while it enables faster wire traversal.

The total interconnect capacitance is the sum of ground and coupling capacitances, C_{GND} and C_C. The effective C_C depends on the switching behavior of adjacent wires, which is characterized by the Miller Coupling Factor (MCF) in Equation 3.1:

$$C_{WIRE} = C_{GND} + 2 \times MCF \times C_C \tag{3.1}$$

The value of the MCF parameter depends on the relative switching activity between two adjacent wires. The effect is illustrated in Figure 3.5 for two neighbouring wires, X and Y. When both wires switch to the same direction, the MCF parameter is 0 and the total wire capacitance is only C_{GND}. On the contrary, coupling capacitance is maximized (MCF of 2) when the wires switch in the opposite direction. When only one of the wires switches, then MCF is 1 and the total capacitance equals $C_{GND} + C_C$. It is possible to reduce coupling capacitance by increasing the wire spacing, or by introducing shielding (interleaving constant VDD/GND wires and data wires), but this comes at the cost of an area penalty. Wire spacing effectively reduces the value of C_C, while shielding decreases MCF. A key challenge in interconnect design is to reduce either C_C, or the worst-case MCF, while maintaining the same physical footprint of the interconnect, thereby reducing the effective wire capacitance, the delay, and energy consumption.



X	Y	C_{EFF}	MCF
_/‾	___	$C_{GND}+C_C$	1
_/‾	_/‾	C_{GND}	0
_/‾	‾_	$C_{GND}+2C_C$	2

Figure 3.5: The values of the Miller Coupling Factor (MCF) and thus, the total effective capacitance for two adjacent wires, X and Y, depending on their relative switching activity.

In the new NoC design proposed in this paper, wire capacitance is significantly reduced by exploiting the out-of-phase operation of the two unidirectional links that connect two adjacent routers. Assume, for example, the routers A and B shown in Figure 3.6. The A$\rightarrow$B link is used to transfer data from router A to router B. Data is launched on the positive edge of the clock and captured on the negative edge of the clock by the input buffers of router B. On the contrary, data from B to A is transferred on the other half-period, starting from the negative edge and ending on the next positive edge. In this case, when one direction is activated in one half of the clock cycle, the other one remains idle, waiting for the other half to send its data (the timing diagram of Figure 3.6 depicts this behavior).

Based on this property of link activation in different clock phases, we can employ wire interleaving in the physical layout. The wires of the two uni-directional buses connecting adjacent routers are interleaved, as shown in Figure 3.7. This pattern eliminates the undesired MCF=2 of the coupling capacitances of neighboring wires.

Wire-interleaving prohibits adjacent wires from switching in opposite directions, thus **achieving a worst-case MCF of 1 (rather than 2)**, *inde-*

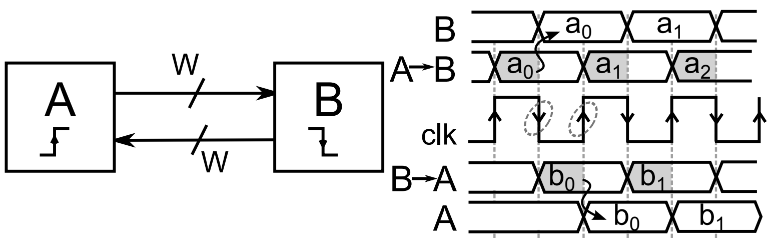


Figure 3.6: Since flits are launched and captured on alternate clock edges, only one of the two inter-router links is active in each half-cycle.

pendent of the switching activity profile of the transmitted data. Bit i of the link from A to B is routed between bits i and $i-1$ of the link from B to A. Since the two links are never active in the same half-cycle, when bit i of the A→B link makes a new transition, it is guaranteed that its adjacent bits – that belong to the B→A link – are "quiet," since they have completed their transitions in the previous half-cycle. The adjacent wires can be considered as active shields for the bit that changes its value. The same situation occurs in the next half-cycle, which drives the wires of the opposite direction. Overall, in any half-cycle, every bit will make a transition with *the guarantee that its neighbors remain constant*. Due to this property, MCF cannot exceed 1, thereby offering a significant reduction in the effective wire capacitance. This guarantee holds for all links of the NoC in either direction, which provides significant overall energy savings. As proven in [119], the energy savings in the links when limiting the largest value of MCF to 1 (as opposed to 2) is equal to:

$$E_{saving} = 1 - \frac{C_{WIRE(MCF=1)} V_{DD}^2}{C_{WIRE(MCF=2)} V_{DD}^2} = 1 - \frac{C_{GND} + 2C_C}{C_{GND} + 4C_C}$$

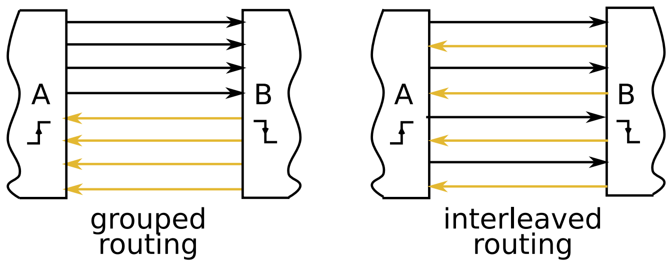


Figure 3.7: Wire-interleaving of half-cycle links limits the worst-case MCF to 1 (rather than 2), independent of the data-switching activity profile.

The energy savings range between 25% and 40%, depending on the value of C_C, relative to C_{GND}, which depends on the metal layer and the wire geometry and spacing [119]. Moreover, note that the lower wire capacitance makes link traversal even faster, thus making the half-cycle requirement easier to achieve for the scaled inter-tile links.

3.4 Router Buffer Reduction

Half-cycle links change the notification loop of credit-based flow control, since data is sent in the forward direction on one edge of the clock, while credit updates are returned in the reverse direction on the opposite edge of the clock. This effectively *reduces the RTT by one cycle*. To enjoy the decreased RTT, the credit updates that arrive at the middle of the cycle must be reused immediately. In this way, if credit updates are not handled appropriately, combinational paths launching in the middle of the cycle are created. These could potentially degrade the router's speed. To avoid this negative effect and still enjoy the reduced RTT and its associated buffer-saving properties, we redesign the Switch Allocation (SA) stage of the NoC routers.

In fast VC-based router architectures, such as [82], the SA stage receives only *qualified* requests. A request is qualified to participate in SA, as long as it refers to an output VC that has available credits. If credit-checking is not performed before-hand, then a grant may be given to an input VC that cannot actually use it, thus leaving the selected output port idle. In fast implementations [82], VC allocation occurs through SA: once a head flit is assigned to an output port, it also receives an available output VC for that output port.

In the proposed architecture, we employ two SA units, the Primary SA (*pSA*) and the Oblivious SA (*oSA*) units, placed in parallel, as shown in Figure 3.8. Being the primary unit, *pSA* performs the normal SA operation, including credit checking. The *oSA* unit performs, in parallel, the same task (albeit in a simplified manner), but it only accepts the requests of the inputs (or input VCs) that refer to outputs (or output VCs) without any credit available.

When an output VC has available credits and an input VC is requesting it, then this resource will definitely be utilized. In this case, handling the credits returning in the middle of the cycle is not critical. The half-cycle credit updates become critical when the requested resource appears to be unavailable on one edge of the clock, but becomes available on the next opposite edge of the clock (recall that neighboring routers operate

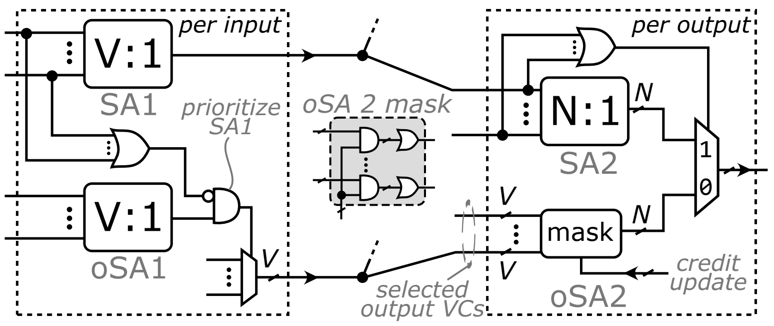


Figure 3.8: The proposed architecture employs two switch allocation units (one primary and a simplified oblivious one), in order to allow any credit updates arriving at the middle of the cycle to be reused immediately. Parameters N and V represent the number of router input ports and the number of VCs per port, respectively.

on alternate clock edges). For this case, the oSA unit allows one request from each input port to qualify for an output request, as long as it refers to an *unavailable* output VC. The request can only eventually be granted if the specific output VC becomes available in the middle of the cycle, through a credit update. In any case, the *pSA* unit's decisions are always prioritized over *oSA*'s.

The operation of *oSA* is split in two stages, as shown in Figure 3.8. In the first stage (*oSA1*), each input port independently selects one input VC headed to an unavailable output VC. The *oSA1* winners reach their destined output port, where at most one winner is determined in *oSA2*. At this point, half the critical path has been traversed, and, thus, the credit-update signals have arrived. Since only one credit update for a single output VC may arrive for each output port, only the input request that refers to that output VC can be granted. All other requests refer to output VCs that are still unavailable and are discarded through a masking process, without the need for any arbitration. Thus, the *oSA2* stage is a simple masking process, unlike the more complicated second-stage arbitration process typically found in SA units. Note that the only extra logic added by *oSA* can be seen in the lower half of Figure 3.8; the upper half illustrates the organization of a baseline SA.

In this way, the incoming credit-update signal is used with no delay cost, and this enables immediate credit consumption, which effectively **lowers the credit notification loop by one cycle**. One half-cycle is saved in the forward data direction, and one half-cycle is saved in the backward (credit-update) direction. Reducing the RTT by one cycle translates to a saving of one buffer slot per VC. In the case of skewed traffic, each

VC (independently of the rest) should have as many buffer slots as the link's RTT, in order to ensure full-throughput of data transfers.

3.5 Experimental Evaluation

In this section, we evaluate the performance of the proposed NoC architecture and compare it – in terms of hardware complexity and network performance – with baseline NoC architectures that assume full-cycle link traversal.

As previously explained, the two fundamental contributions of the new design are: (1) the link power reduction, and (2) the buffer size reduction (thereby leading to further overall power reduction). We begin our evaluation by assessing the improvements in power consumption obtained when using the proposed architecture.

Figure 3.9 depicts the normalized power of an 8×8 2D mesh baseline NoC with full-cycle links ("Base"), as compared to the proposed design with half-cycle links ("Proposed"). Both designs operate at 1 GHz, support 2, 4, and 6 VCs per port, and the NoC routers (with 5 input/output ports, as needed by the 2D mesh topology) are connected with 2 mm 64-bit links (following the floor-planning sizes of [120]). In each configuration, VC buffers include 3 slots/VC for the baseline case and 2 slots/VC for the proposed architecture, which are enough to cover the RTT in each case. All NoC designs under investigation were implemented in VHDL, synthesized, and placed-and-routed using a 45 nm 0.8 V standard-cell library.

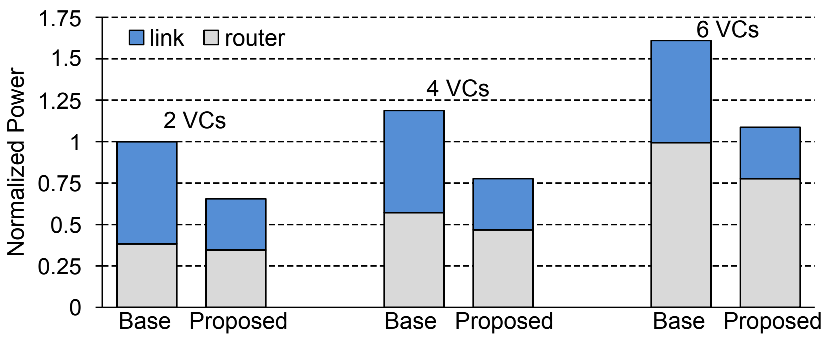
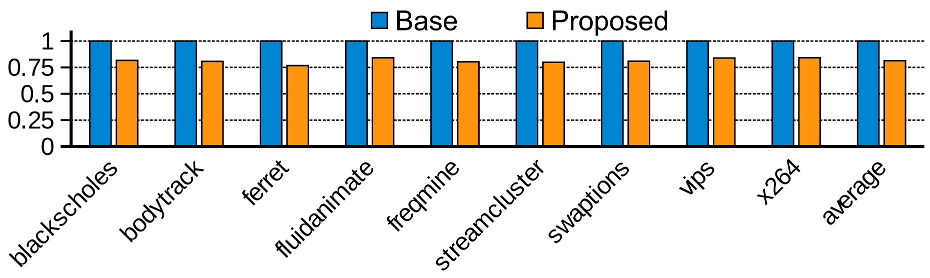


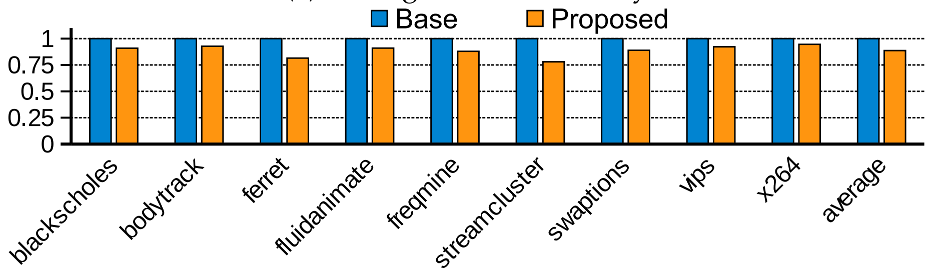
Figure 3.9: The normalized power consumption of the proposed architecture, as compared to a baseline NoC with full-cycle links. Power savings are due to both link- and buffer-power reductions.

From the obtained results, measured under equal NoC utilization (throughput), we observe that the savings in both the link and buffer power consumption (due to reduced wire capacitance and smaller buffers, respectively) offered by the new design translate to significant overall NoC power savings of 34%, on average. It is evident that the contributions of both the link optimizations and the buffer reduction are significant for all examined VC configurations. Specifically, link power reduction accounts for 75%, on average, of the total power savings, with the remaining 25% being attributed to the buffer power reduction.

Besides power reduction, the proposed router design also achieves small area savings, mostly due to the buffer reduction, which outweighs the small area added by the second SA unit. Recall that the latter is added to handle the credit updates arriving at the middle of the cycle, without any delay penalty.



(a) Average network latency

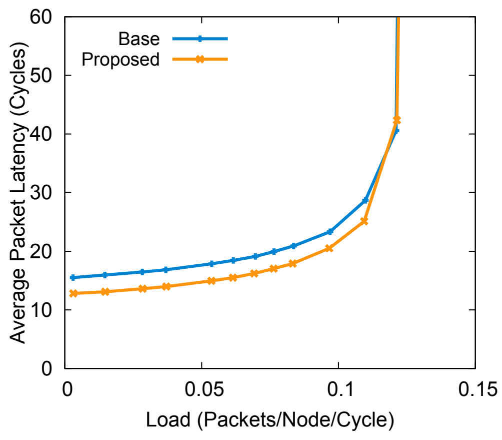


(b) Total execution time

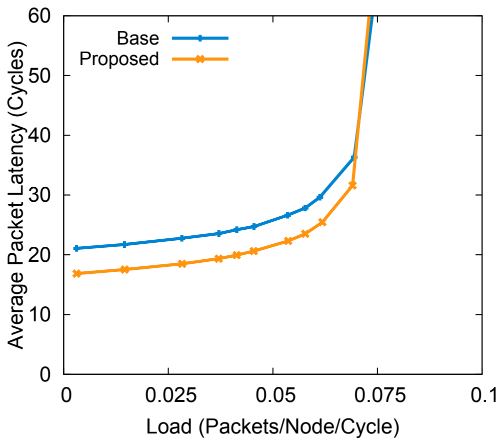
Figure 3.10: Full-system, execution-driven simulation results: (a) average network latency, and (b) total execution time of various PARSEC benchmark applications [20] running on a 64-core CMP.

Note that the results obtained here (i.e., at 45 nm and assuming a 2 mm inter-tile distance) can also scale to smaller technologies, assuming (a) a diminishing tile area, and (b) slowly increasing clock frequencies. As previously mentioned, these two assumptions are already part of the prevailing tendency in current CMP design practices.

Having established the substantial power reduction enabled by the proposed design, it is now imperative to assess its impact on overall system performance. Toward this end, we simulate a 64-core tiled CMP system running real application workloads. The system is configured according to the parameters of Table 1.1, with 3 VCs per input port and assuming half-cycle link delays. Figure 3.10 shows the average network latency and the total execution time of various PARSEC multi-threaded benchmark applications [20], normalized to a baseline NoC with full-cycle links. The proposed architecture offers significant average network latency savings, which range from 16% to 23% (19%, on average). This latency reduction translates to a direct reduction in the total execution time, ranging from 5% to 22% (11%, on average). Most importantly, such savings are achieved while still enjoying significant power reductions.



(a) Uniform Random (UR)



(c) Bit-Complement (BC)

Figure 3.11: Latency vs. load curves under UR and BC traffic patterns for baseline NoCs with full-cycle links and RapidLink architectures with half-cycle link traversal.

Despite the authenticity provided by execution-driven, full-system simulations, the flexibility to stress the NoC is somewhat limited, due to the fixed characteristics of the running applications. Hence, we also employ synthetic traffic patterns Uniform Random (UR) and Bit-Complement (BC) traffic in our evaluation. Figure 3.11 depicts the average network latency versus input load of a baseline NoC with full-cycle links and a NoC using the proposed architecture with half-cycle links. The latter achieves, on average, 18% and 20% lower latency under UR and BC traffic, respectively, as compared to the baseline NoC. In terms of throughput, both designs achieve equal performance. The positive effect of half-cycle link traversal is more pronounced under traffic patterns with increased average zero-load latency (i.e., larger average hop count).

Hence, the substantial power savings reaped by the proposed new design come with no negative impact on network performance. On the contrary, our evaluation indicates substantial performance gains, in terms of network latency and real-world application performance.

3.6 Conclusions

Tile-based homogeneous CMPs are characterized by a regularity in their physical layout. As technology scales, the number of on-chip tiles increases, while the size of each tile decreases. Consequently, the inter-tile distances scale *down* proportionally to the decreasing tile dimensions. In this paper, we exploit decreasing inter-tile wire lengths to facilitate ultra-fast NoC link traversal in half a clock cycle. This feat is achieved using appropriate wire engineering techniques, and by utilizing both edges of the clock during the sending/receiving of flits. The proposed NoC design leverages half-cycle link traversal to reap substantial power savings. The savings are derived by commensurate reductions in (a) link power (irrespective of the data-switching patterns), and (b) buffer power (by reducing the buffer size). Most importantly, the power benefits do not incur any performance penalty; on the contrary, the latency is actually improved. Extensive hardware implementation analysis using placed-and-routed designs validates the efficacy of the proposed architecture, which yields significant power savings of 34%, on average. Additionally, cycle-accurate full-system simulations using real benchmark applications also demonstrate significant performance improvements, i.e., average reductions in network latency and total execution time of 19% and 11%, respectively.

Chapter 4

Networks-on-Chip with Double Data Rate Links

The need for higher throughput and lower communication latency in modern Networks-on-Chip has led to low- and high-radix topologies that exploit the speed provided by on-chip wires - after appropriate wire engineering - to transfer flits over longer distances in a single clock cycle.

In the previous chapter we exploited the asymmetry between the delay of the routers and the inter-router links on tile-based CMP interconnects, to achieve half-cycle link traversals, thus reducing link and buffer power. In this work, we rely on the same property, and propose the *RapidLink* architecture, that transfers flits between two adjacent routers connected with links of reasonable length in *half a clock cycle* and *utilizes both edges of the clock* during the sending and receiving of flits. As a result, each upstream/downstream router pair benefits from Double-Data-Rate (DDR) transfers, whereby two flits can be sent/received per clock cycle.

In RapidLink, the original clock frequency of the NoC is unaffected, and the NoC routers do *not* need to run any faster than their normal operating frequency. The only constraint is that the Link Traversal (LT) delay cannot exceed one half of the delay of the router, which is feasible for small/medium wire lengths, after appropriate wire engineering, as elaborated in Section 3.1. However, RapidLink can also handle the cases where the link delay *cannot* fit within half of a clock cycle. In those cases, RapidLink "fragments" the link into multiple DDR half-cycle segments using newly introduced novel DDR **Dual-Stream Elastic Buffers (DS-EBs)**, which act as flow-controlled DDR pipeline registers. In this way, the benefits of DDR link traversal can be reaped by *any* NoC design, even those with long wires and increased link delays.

The RapidLink architecture and the new DDR DS-EBs are exploited to provide multiple NoC configurations that can be applied to NoC designs supporting Virtual Channels (VC), or to simplified single-stream (aka wormhole) NoC designs. In all cases, the adoption of RapidLink can markedly decrease latency and increase throughput, without incurring any additional hardware cost, as verified by extensive cycle-accurate network simulations, and detailed hardware analysis.

4.1 RapidLink NoCs with DDR Links

RapidLink aims at providing Double-Data-Rate transmissions on the NoC links *without* constraining the delay of router traversal, which should remain a full-cycle operation. When the NoC links operate in DDR mode, it means that the sender and the receiver on each link should be able to send and receive flits at both the positive and negative edges of the clock. To enable this operation, the NoC routers can provide two separate send/receive paths to each inter-router link, where each path carries a **separate *stream (flow) of data***. With RapidLink, these two separate streams (data flows) can be transferred in a *time-multiplexed* manner across the same inter-router link in DDR mode; one stream would "ride" the positive phase of the clock, while the other stream would "ride" the negative phase.

Each router in the baseline RapidLink architecture consists of *two full-cycle wormhole sub-routers* of W-bits each, as depicted in Figure 4.1. The output of the two sub-routers is time-shared on a W-bit inter-router link using both edges of the clock. For half a clock period (e.g., during the positive phase of the clock), the link is traversed by flits of sub-router 0, while during the other half of the clock period (e.g., the negative phase), flits of sub-router 1 use the link. The two sub-routers operate locally on opposite clock edges and they connect to sub-routers with opposite clock edges downstream, due to the half-cycle link traversal.

The output data of each sub-router is fed into a shared Double-Edge Triggered (DET) register. Each DET register consists of two latches placed in parallel, which are enabled on opposite phases of the clock, and an output multiplexer driven by the clock signal [132]. Thus, DET registers incur marginal overhead, as compared to generic single-edge-triggered registers, which are also built using two latches (master and

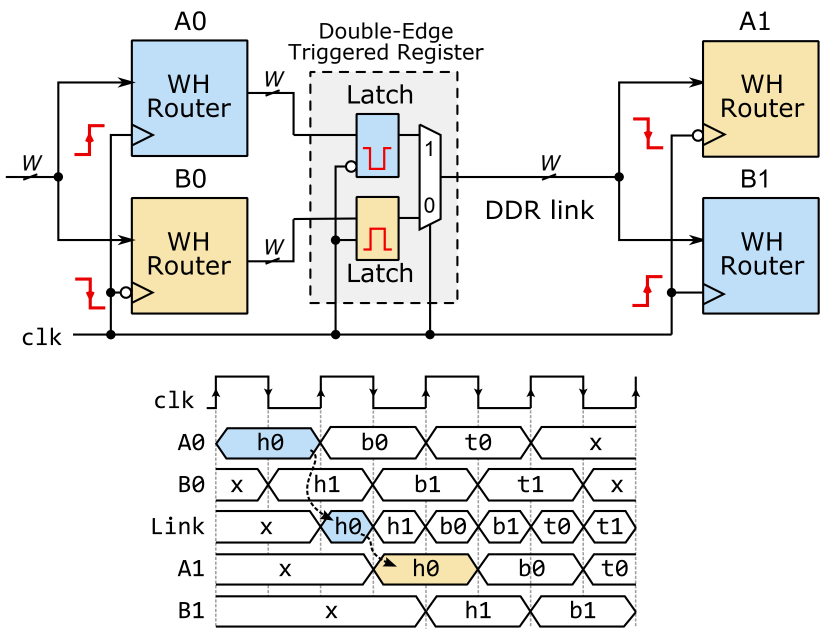


Figure 4.1: RapidLink employing two full-cycle routers and DDR links. Each node consists of two W-bit wormhole (i.e., single-stream) sub-routers that serve separate network streams (e.g., request and response message classes), and they time-share a W-bit inter-router link using both edges of the clock.

slave latches placed in series). The clock signal driving the multiplexer of the DET register is appropriately gated when no new valid flits arrive from any of the sub-routers, thus preventing unnecessary switching activity on the DDR link.

Figure 4.1 also illustrates the activity on the link, as flits flow from routers A0/B0 to routers A1/B1. On the positive edge of cycle 0, flit 'h0' enters router A0, while, on the negative edge of the same cycle, the 'h1' flit is written in the input buffer of router B0. On the next positive edge (cycle 1), the A0 latch stores flit 'h0,' which has completed a whole cycle inside the router and moves directly to the link. Half a cycle later, the flit reaches router A1 and is captured on the positive clock edge. At the same time, 'h1' appears on the link, and the same pattern continues in the following cycles.

By construction, RapidLink serves two streams of data that traverse the network in a time-multiplexed fashion using distinct sub-routers of alternating clock edges. Thus, effectively, RapidLink implements two distinct sub-networks that remain in isolation, i.e., moving data from the first sub-network to the second sub-network is impossible, due to the physical separation imposed by clock-edge interleaving. Equivalently, this organization resembles a NoC with two Virtual Channels (VCs), where the multiplexing of the flits of each VC is statically determined by the phases of the clock, and exchanging traffic across VCs (i.e., allowing the packet of one VC to transfer to another VC) is not possible[1].

Therefore, the effectiveness and the hardware complexity of RapidLink should be judged relative to a network that hosts two VCs in one network and comprises VC-based routers (supporting 2 VCs) with inputs and outputs of W-bits. The experimental results presented in Section 4.3 verify that the combined area/power budget of the two sub-networks of RapidLink (i.e., two sub-routers per node) is lower – or equal to – the area/power of a network that supports two VCs. This property is attributed to the fact that the two sub-routers of RapidLink are both faster – due to the simplification of parts of the allocation and multiplexing logic – and more area-efficient under equal delay.

4.1.1 Dual-Stream Elastic Buffers (DS-EB)

The data that travels on the DDR link should reach the next router in one half of a clock cycle. This requirement can, perhaps, be satisfied in scaled *tiled* CMPs, after appropriate wire engineering. However, in the general case, fast wire traversal may *not* always be possible. In such cases, the link should be pipelined using dual-edge triggered registers [119], thereby splitting the link into multiple half-cycle LT steps. Instead of adding mere pipeline registers, we propose a new approach. For the first time – to the best of our knowledge – we propose the use of DDR *elastic* buffers, similar in vein to traditional single-stream, single-data-rate EBs [62, 25, 90]. The newly proposed **DDR Dual-Stream Elastic Buffers (DS-EB)** can both (a) split the timing paths on the link, and (b) enable the flow-controlled transfer of data across routers, effectively converting the pipelined channel into a distributed DDR FIFO queue.

The proposed DDR DS-EB replaces the DET flip-flop at the sub-routers' outputs, in order to control the data flowing through the two streams. It operates under a minimal elastic protocol and can be used in a plug-and-play manner to pipeline the DDR link (e.g., to meet timing constraints) in any number of *buffered* stages. As shown in Figure 4.2, each dual-stream DDR link consists of a data bus and two pairs of handshaking

[1]Completely separating the traffic of different VCs is a useful property in NoCs when VCs are used to implement different message class that need to remain separate (e.g., request/response traffic).

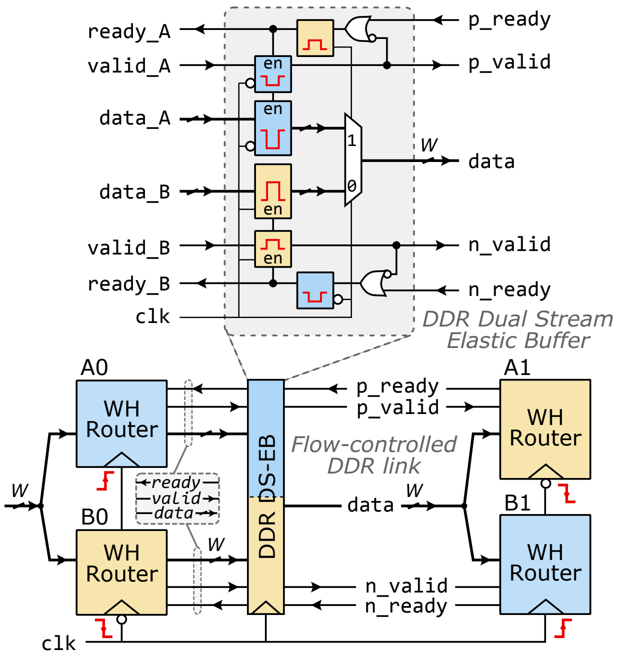


Figure 4.2: The organization of the proposed DDR DS-EB (upper part) and its integration at the output of a RapidLink pair of sub-routers (lower part).

signals (one for each stream), which control data transmissions occurring at the positive and negative levels of the clock. For each stream, a forward valid signal indicates whether the data bus contains valid data, while the backward ready is used by the receiver to indicate when it is ready to accept new data. Whenever valid and ready are asserted, data transmission was successful during that phase of the clock; in all other cases, no transfer occurred.

The organization of the elastic buffer, seen in the upper part of Figure 4.2, uses two independently-enabled data latches, latches for the valid bits, and a multiplexer controlled by the clock signal. Each latch captures the data sent by the two opposite-clock-edged sub-routers. The two latches of each stream are controlled by one more latch that captures the backward ready signal, generated by the input buffer of the downstream router to indicate buffer availability. The latched ready bit acts as an enable signal for the data and valid latches, retaining data whenever the receiving buffer could not capture the last transfer (or, if no transfer occurred in the previous cycle). Note that the receiving sub-router will capture the arriving data on the *opposite* clock edge and, thus, the backward ready signal is captured by a latch enabled by an opposite clock level relative to the corresponding valid and data bits.

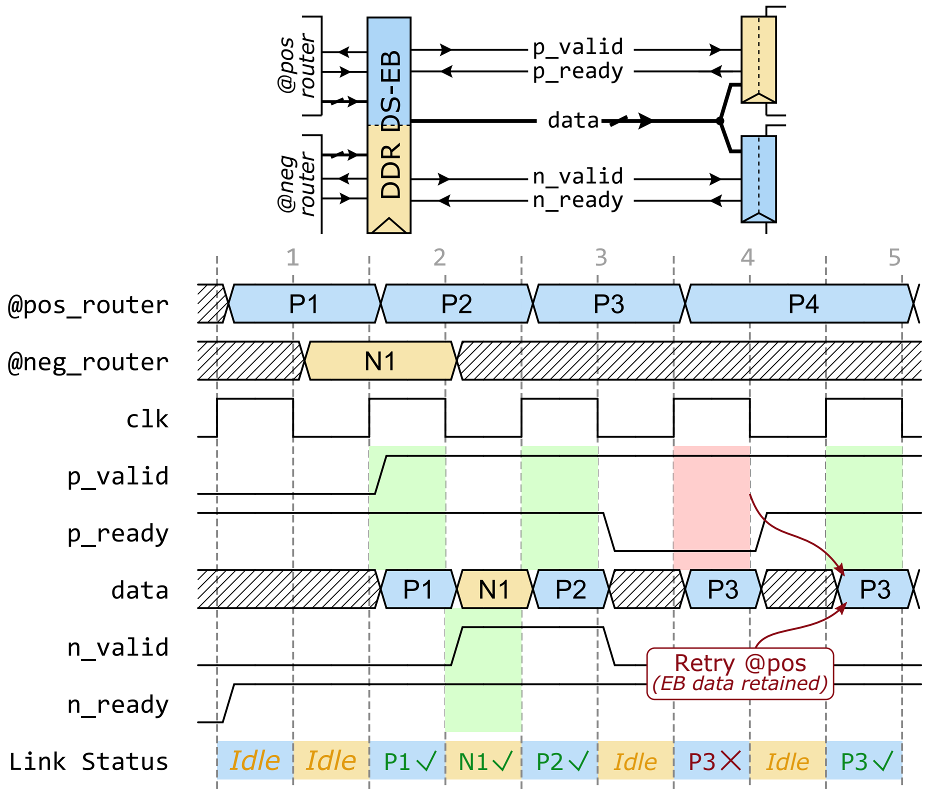


Figure 4.3: Cycle-by-cycle activity of the flow-controlled DDR link. The two data streams P and N are sharing the DDR link in alternating clock phases, and their transfers are controlled by the corresponding ready and valid handshake.

The operation of the DDR DS-EB is demonstrated in the running example of Figure 4.3, which illustrates the cycle-by-cycle activity of a DDR flow-controlled link between two nodes. Two streams P and N are multiplexed on the DDR link. During the first cycle, neither of the two streams is utilizing the DDR link, as both valid signals are low. Flit 'P1' appears in the first cycle at the output of the positive-edged sub-router and reaches the link during the following positive clock phase (p_valid is asserted). The transfer of flit 'P1' was successful, since p_ready was sampled high on the previous falling edge of the clock. During the negative clock phase, flit 'N1' is sent through the link and flit 'P2' is transferred during the following positive clock phase. However, in the next negative phase, the link remains idle, since n_valid is de-asserted. Next, the transfer for the positive stream fails (flit 'P3'), since the receiver's p_ready was de-asserted on the previous falling clock edge. Therefore, the latched ready signal of the DS-EB is preventing the valid and data latches of the P stream to be updated; otherwise, the stored flit ('P3') would be overwritten by the incoming flit ('P4'). In the negative clock phase, flit 'P3' cannot retry its transmission, since this phase is reserved for stream N, and the link remains idle again. Finally, as the clock phase switches, stream P is allowed to retry (p_ready is high) and 'P3' is actually transferred successfully.

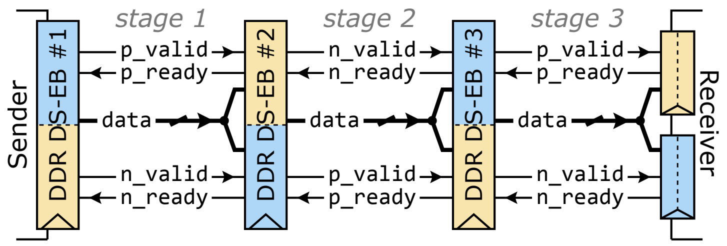


Figure 4.4: Multiple DDR DS-EBs may be placed in series – acting as a distributed DDR FIFO – to pipeline the DDR link. Neighboring DS-EBs should be driven by alternating clock phases.

The DDR link can be pipelined by placing multiple DS-EBs in series, as depicted in Figure 4.4. To achieve half-cycle traversal across each link segment, and full throughput DDR transmission, consecutive DS-EBs should use alternate clock phases.

4.1.2 Integrating RapidLink with Single-Data-Rate Network Interfaces

In RapidLink, the DDR operation of the NoC links does *not* affect the full-cycle (single-edge) operation of the NoC routers, nor the Network Interfaces (NIs). Each NI can safely assume an injection/ejection throughput of at most 1 flit/full-cycle/NI, as in any single-edge-triggered baseline NoC. Although this feature simplifies RapidLink's integration, since no NI modifications are required, it introduces a data rate mismatch between the NIs and the DDR operation of the NoC links. To interface between the two domains with different data rates, while preserving RapidLink's ease of integration, we provide two lightweight *bridge modules* that act as "glue logic" between the injection and ejection points of the network.

Figure 4.5 illustrates the architecture of the bridges used at the injection and ejection points of the network. At the injection points, we should *merge* two independently flow-controlled data streams that operate on

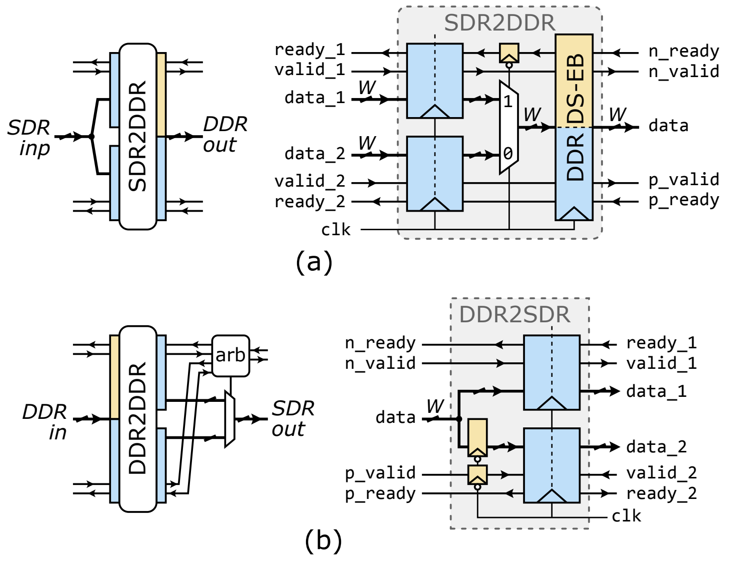


Figure 4.5: (a) The SDR2DDR bridge for stream merging (at the injection point), and (b) the DDR2SDR bridge for stream splitting (at the ejection point). Both bridges connect RapidLink's DDR inputs and outputs to single-rate network sources and sinks, respectively, which can handle messages from two distinct streams (flows).

their own ready/valid handshake – driven on the positive clock edge – to a dual-stream DDR interface. Merging is performed by the SDR2DDR bridge, as illustrated in Figure 4.5(a). The injected flits are first buffered into one of the positive-edge-trigerred elastic buffers, depending on the stream they belong to. From there, they move to the DDR DS-EB that will guide them to the appropriate sub-router of the network, through a multiplexer that is controlled by the phase of clock. In order for the handshake signals to be sampled properly, the `n_ready` signal of the DDR DS-EB is re-timed before being transferred to the positive-edge-triggered EB of the injection interface.

At the ejection points, we need to perform the opposite operation and *split* the two data streams coming out of RapidLink on opposite clock edges, to two independently flow-controlled streams that operate on a single clock edge. This splitting is performed by the DDR2SDR bridge, shown in Figure 4.5(b), which de-multiplexes the DDR stream back to two separate singe-data-rate streams by placing the flits ejected from the network into two parallel positive-edge-trigerred EBs. The positive-edge-trigerred EBs capture data only once per clock cycle, while the DDR input provides new data twice per clock cycle. Therefore, one data item per clock cycle will be lost and not captured by any of the two EBs.

To resolve this issue, we add a re-timing stage for `data` and `p_valid` bits, in order for the data of the P stream to wait safely for the next positive clock edge, and not be overwritten by the new data of the N stream that appears in the middle of the cycle. Finally, once the two streams are separated at the output of the DDR2SDR bridge, they need to arbitrate to gain access to the exit of the network.

4.1.3 Supporting Multiple VCs

The baseline RapidLink design supports the transfer of two independent data streams flowing in separate sub-networks, each one synthesized using simplified wormhole sub-routers, i.e., routers without VC support. Nevertheless, the baseline architecture can be extended to support more than two flows, either for implementing more complex protocols that require more than two separate message classes, or to increase performance.

RapidLink can support V VCs in total, by assigning half of them to each sub-network. A network node consists of two sub-routers, each one hosting $V/2$ VCs, as shown in Figure 4.6, for the case of a 4-VC configuration. In each clock cycle, at most one flit may appear at each sub-router output, according to typical VC-based router architectures [36]. When a flit tries to reach the output, it is stored into one of the $V/2$ DS-EBs that are placed in parallel at every output port. A one-hot valid vector encodes the flit's VC id, indicating the DS-EB where the flit will be stored. For the 4-VC configuration of Figure 4.6, each sub-router serves 2 VCs, providing 2-bit valid outputs to the 2 DDR DS-EBs.

Driving the flow-controlled DDR link with only one flit per clock phase is accomplished by arbitrating among the DS-EBs that currently host valid flits. Two $V/2$-input arbiters serve the two streams and determine which one of the $V/2$ VCs can use the link during the positive and negative clock phases, respectively. Arbiter requests are only made by the VCs whose associated DS-EB valid bits are asserted. As shown in Figure 4.6, the select signal of the data multiplexer used to drive the link is the corresponding arbiter's grant, depending on the current phase of the clock. Data reaches the receiver, along with a one-hot valid vector per stream that indicates the input VC buffer where the flit must be stored within the downstream router. Similarly, each downstream sub-router generates a $V/2$ ready vector to indicate buffer availability of each VC buffer. At the end of the active clock phase, if the granted VC's

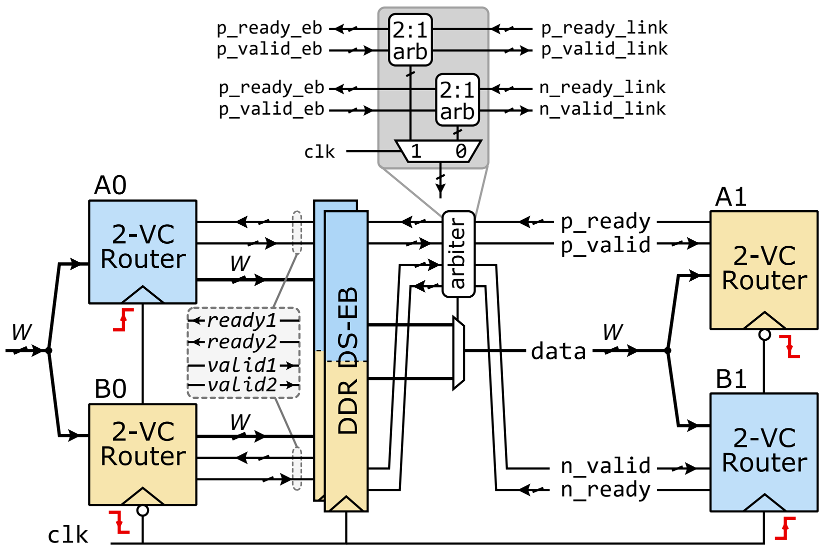


Figure 4.6: The RapidLink organization supporting a total of $V = 4$ VCs. Multiple DS-EBs are placed in parallel at the output of the RapidLink sub-routers to support $V/2$ VCs per clock-phase.

incoming ready signal was asserted, the transmission was successful. Otherwise, the stream can retry in its next clock phase.

4.2 Single-Stream RapidLink

The baseline RapidLink architecture supports, by construction, the transfers of two independent data streams on the NoC links. In most systems, supporting two independent data streams is a necessity, since the system operation is based on a higher-level transaction protocol that requires the use of separate message classes. These message classes can range from simple request/response traffic to ones encountered in more complex cache-coherence protocols. However, if multiple message classes, or VCs, are *not* required, we need to identify a way to enable RapidLink to operate on *a single message stream per network source*, even if it is inherently constructed to support two message streams per source.

4.2.1 Concentrated RapidLink

Each network source of RapidLink, as shown in Figure 4.5(a), injects in the network two independent message flows that are multiplexed inside RapidLink on opposite clock edges. Equivalently, we can replace the two message flows of one source with the traffic originating from two

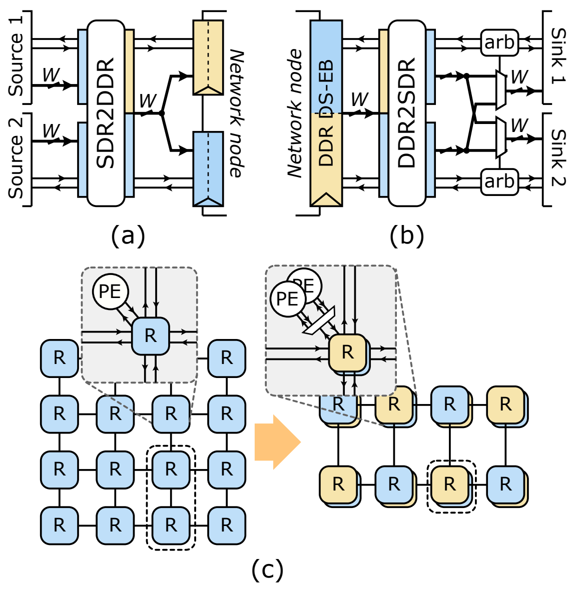


Figure 4.7: (a) The injection and (b) ejection network interfaces that allow two single-stream sources and sinks to be attached to a *concentrated* RapidLink node. (c) A 4×4 2D mesh (left) is folded onto a 2×4 mesh (right) with DDR links, implemented with 5-port RapidLink routers, where each sub-router serves two Processing Elements (PEs).

single-message-flow sources. To achieve this, we connect to the input of an SDR2DDR bridge two network sources, as shown in Figure 4.7(a). This network-source merging in the SDR2DDR bridges is equivalent to network concentration [77], where two source/sink nodes of the original network are mapped to a single RapidLink node.

In this way, by time-multiplexing the two concentrated flows on DDR links, the network diameter is effectively reduced, without any implementation overhead, since neither the router population, nor the router radix are increased. An example of the method's applicability is presented in Figure 4.7(c) in a 4×4 2D mesh serving 16 Processing Elements (PEs; e.g., CPUs), with each one connected to the network through a single in/ejection port. In a concentrated RapidLink (right-hand side of Figure 4.7(c)), each vertical pair of nodes is merged into one that operates in DDR mode. This leads to a "folding" of the original single-data-rate 4×4 2D mesh onto a 2×4 mesh with DDR links.

Equivalently, at the ejection points, shown in Figure 4.7(b), the DDR RapidLink stream is split in two flows using the DDR2SDR bridge. Since we cannot guarantee that the two split flows are not destined to the same output, an extra arbitration and switching stage is employed prior to ejection, in order to guarantee that only one stream can have access to each of the two connected sink points.

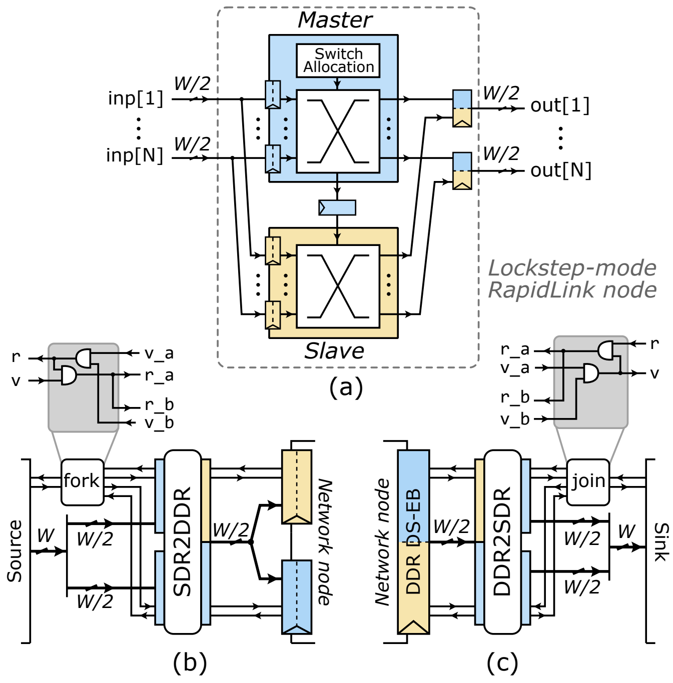


Figure 4.8: (a) The organization of a RapidLink node that employs master and slave sub-routers operating in *lockstep mode*. The organization of the (b) injection and (c) ejection interfaces that split the packets of a single stream to two independently flow-controlled streams driven on opposite clock edges.

4.2.2 Lockstep-mode RapidLink

Instead of merging two single-stream network sources to produce the dual DDR stream required by the baseline RapidLink (as described in the previous sub-section), we can produce the two streams by *splitting the packets of a single-stream in half*. In this configuration, flits entering the network are split in two, and the two halves (of $W/2$ bits each) are injected in consecutive opposite clock edges into the two narrower sub-networks.

This organization is shown in Figure 4.8. The two halves of the same packet travel “tied” together, appearing on the DDR links one after the other on consecutive (alternating) clock levels. To achieve this, the two sub-routers make sure that only one of the two halves is competing with other flows, arbitrating and allocating resources; the other one, always follows. To achieve this behavior, the two sub-routers are not treated equally. The "master" sub-router preserves the original switch architecture, including routing and allocation logic, and is responsible to perform arbitration among contending flows. The "slave" sub-router operates in *lockstep mode* with the master, and routes flits in *exactly the same way* as the master, by blindly copying its arbitration decisions. The switch configuration is transferred to the slave through a re-timing stage. In this way, a whole cycle is provided to the full-fledged router (the master), while a *half-cycle* path is enforced to the slave router, which only involves simpler multiplexing circuits.

The master and slave sub-routers must be set to alternating clock edges on consecutive nodes, so that the pre-pending flit always appears first on each node. Interfacing with the network requires re-wiring the W-bit input of the packet sources to the two $W/2$-bit streams (on injection) and back (on ejection), as shown in Figs. 4.8(b) and (c), respectively. Note that the injection wiring must make sure that the half containing the header (e.g., flit type, destination, etc.) "rides" the clock phase that will follow the master sub-network. Between the source and the SDR2DDR bridge, extra control logic ("fork") is inserted in the flow control signals, to make sure that both flit halves are injected simultaneously, without any idle cycles in-between. At the ejection points, similar logic ("join") guarantees that the valid signal towards the sink is only asserted when both $W/2$-wide outputs are valid.

4.3 Evaluation

In this section, we evaluate the performance of RapidLink and compare it, in terms of network performance and hardware complexity, with baseline NoC architectures that assume full-cycle and single-data-rate link traversal. In the first set of experiments, we evaluate RapidLink for networks supporting two and four VCs. In both cases, routers employ the combined allocation policy [103], and ElastiStore buffers [116] with 3 flits/VC, as needed to cover the credit Round-Trip Time (RTT) in single-cycle routers with full- or half-cycle links.

The performance evaluation involves four synthetic traffic patterns: Uniform Random (UR), non-uniform Localized (LC) traffic, and two versions of permutation traffic: Bit-Complement (BC) and Transpose (TS) traffic patterns. For the latency-throughput analysis, we assume a bimodal distribution of packets with 50% of the packets being short, 1-flit

	Base	RapidLink	RapidLink-Full-LT
2 VCs	1.53 mm^2	1.46 mm^2	1.60 mm^2
4 VCs	3.07 mm^2	3.08 mm^2	3.21 mm^2

Table 4.1: Area occupied by full 8×8 64-bit meshes with 2 and 4 VCs, using the Baseline network architecture and RapidLink with half-cycle and full-cycle links.

packets, and the rest being long, 5-flit packets.

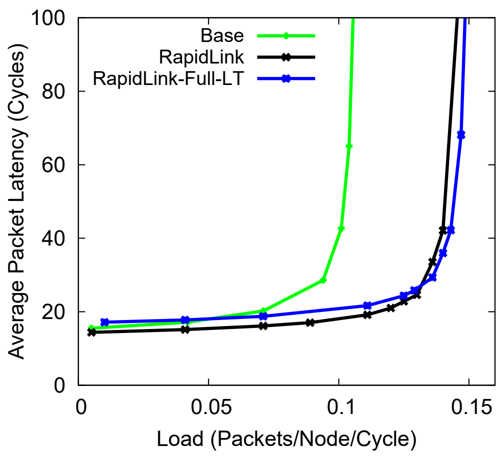
The baseline RapidLink with DDR links creates two independent streams that can be interleaved on the link using opposite edges of the clock. This allows for DDR operation on the links without increasing the clock frequency of the partitioned NoC routers. The DDR link operation is expected to offer up to 2× the saturation throughput of the baseline full-cycle and single-data-rate NoC, while still reducing zero-load latency.

This behavior is, indeed, verified by the network performance results shown in Figure 4.9, which include baseline and two forms of RapidLink DDR NoCs: "RapidLink" assumes half-cycle link traversals, while "RapidLink-Full-LT" assumes full-cycle link traversal that is split in two half-cycle DDR segments using DS-EBs in the middle of the link (as described in Section 4.1.1). RapidLink configurations achieve a 1.3–2× throughput increase under UR, BC, and TS traffic. In the case of LC traffic, saturation throughput is not limited by the NoC inter-router links, thus limiting throughput improvement to 17%. With RapidLink's half-cycle link traversals, latency is also improved by 9% on average. When the DS-EB is used to pipeline the link, latency becomes equal to the baseline network's, but the throughput benefits are fully preserved, due to the DDR operation of the links.

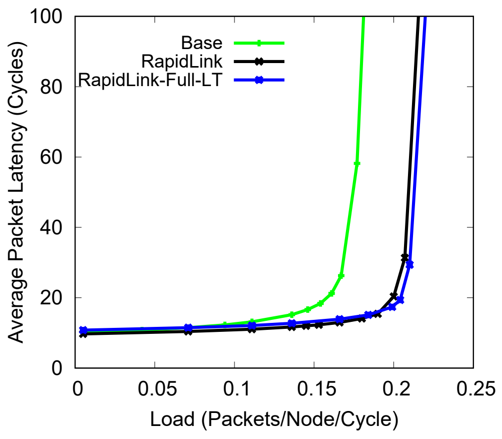
Similar behavior is observed when the NoC supports more than 2 VCs. The results in Figure 4.10 highlight the NoC utilization increase under UR and TS traffic with increasing traffic load, for the case of NoCs with $V = 4$ VCs. As explained in Section 4.1.3, RapidLink utilizes – in parallel – two sub-networks of simpler routers (with $V/2 = 2$ VCs), thereby allowing for significantly higher saturation throughput.

The significant throughput increase offered by RapidLink is achieved without dedicating more resources to the NoC than the baseline NoC with full-cycle and single-data-rate links (neither within the routers, nor on the links).

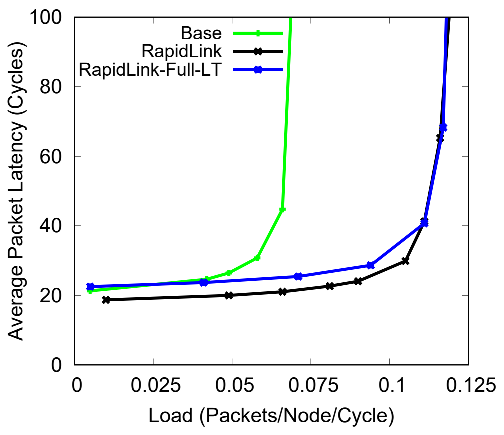
Table 4.1 reports the layout area occupied by a complete RapidLink NoC



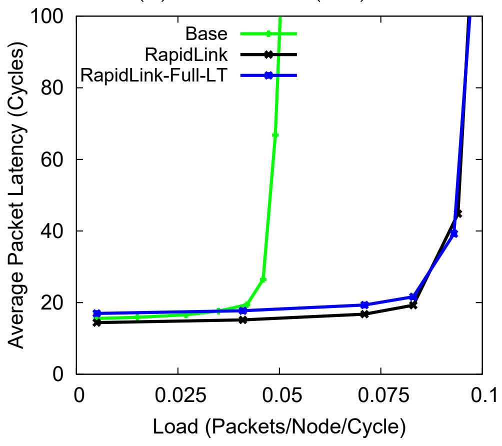
(a) Uniform Random (UR)



(b) Localized (LC)



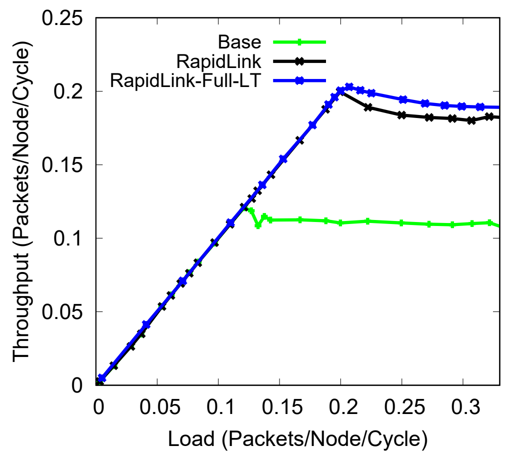
(c) Bit-Complement (BC)



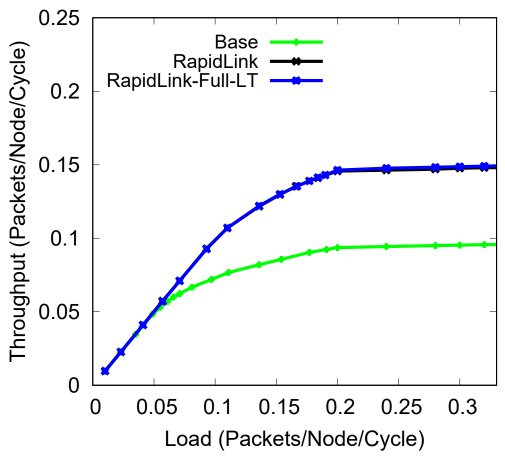
(d) Transpose (TS)

Figure 4.9: Latency vs. load curves for 2-VC networks under UR, LC, BC, and TS traffic patterns. Comparisons include a baseline NoC built from routers with 2 VCs and full-cycle links, and RapidLink NoCs with DDR links that assume full and half-cycle link traversal.

consisting of 2 parallel sub-routers per node with $V/2$ VCs and 64-bit input-output ports, as shown in Figure 4.6, and the area of a NoC with V-VC baseline routers with 64-bit datapaths. Both designs are sized to operate at 1 GHz. When the NoC serves only 2 VCs ($V = 2$), then the two sub-networks of RapidLink do not employ VC- based routers, since they only need to serve 1 VC each. Instead, each sub-network employs simpler wormhole routers, which save both area and reduce the delay, as shown in Figure 4.2. Note that the results of RapidLink also include any bridge modules put in front of the network interfaces, as well as the DS-EBs put on the links. The areas of the baseline and RapidLink designs are almost the same in all examined cases, with RapidLink being



(a) Uniform Random (UR)



(b) Transpose (TS)

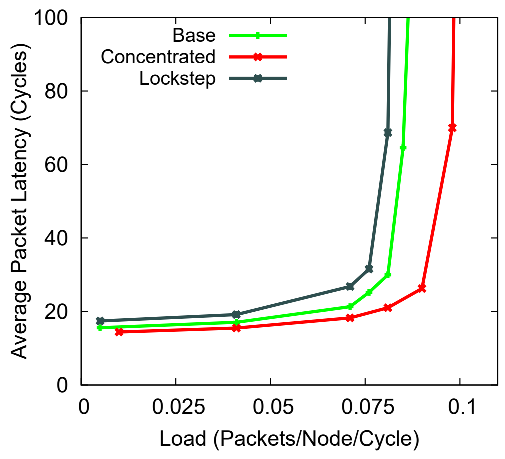
Figure 4.10: Throughput vs. load curves for 4-VC networks under (a) Uniform Random and (b) Transpose traffic patterns, for a baseline NoC, and RapidLink NoCs that assume full and half-cycle link traversal.

slightly larger (less than 5% difference in the worst case). Therefore, the reduced latency and increased throughput provided by RapidLink's DDR link operation is achieved *with negligible hardware overhead*.

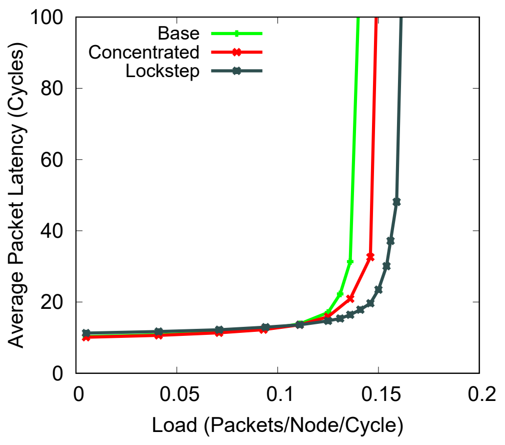
It should be stressed that it is always possible to trade off the network performance benefits for reduced power, by reducing the NoC's clock frequency down to the point that the throughput of RapidLink configurations equals the throughput of the baseline NoC. This would reduce the power consumption of both the clock tree and the routers, while still delivering performance similar to that of the baseline NoC.

As a final step, we compare the performance and hardware complexity of *single-stream* RapidLink configurations (as described in Section 4.2) versus a simple wormhole network that supports – by default – single-stream network sources. Figs. 4.11(a) to (d) depict the latency-throughput curves as functions of the node injection rate, for the four examined synthetic traffic patterns. In each diagram, we include the performance of a baseline wormhole network, of a *concentrated* RapidLink (Section 4.2.1), and a RapidLink NoC with routers operating in *lockstep mode* (Section 4.2.2). Both RapidLink alternatives employ full-cycle link traversals, through pipelining with DS-EBs. In this way, RapidLink imposes the same timing constraints on the links as the baseline wormhole network.

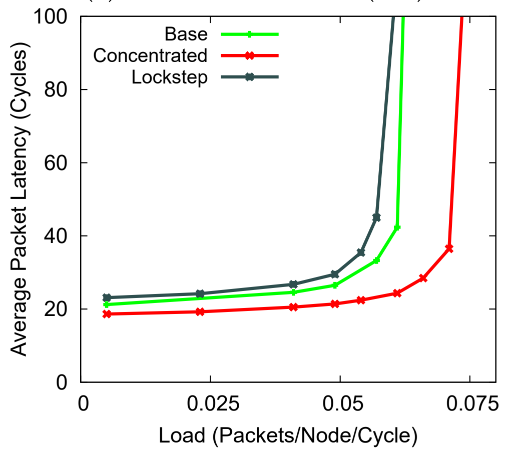
The RapidLink design operating in lockstep mode is slightly worse in



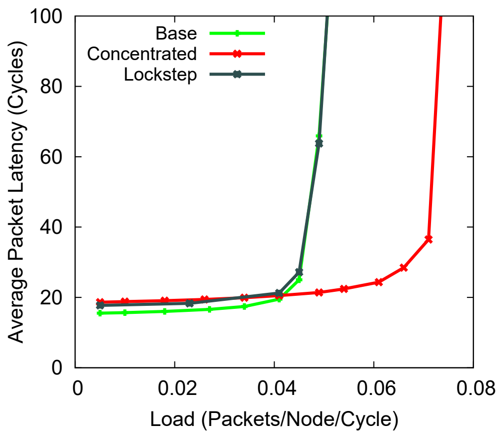
(a) Uniform Random (UR)



(b) Localized (LC)



(c) Bit-Complement (BC)



(d) Transpose (TS)

Figure 4.11: Latency vs. load curves for *single-stream* networks (i.e., with no VCs) under UR, LC, BC, and TS traffic patterns. Comparisons include a baseline wormhole NoC with full-cycle links, a *concentrated* RapidLink, and a RapidLink NoC with routers operating in *lockstep mode*. Both RapidLink alternatives employ full-cycle link traversals, through pipelining with DS-EBs.

terms of throughout and latency than the baseline wormhole network. This small degradation in performance is attributed to a slightly increased latency in the network interface, relative to simple wormhole networks. On the contrary, the concentrated RapidLink offers the highest performance, allowing for 22% higher saturation throughput, due to the increased utilization enabled by the DDR multiplexing of two independent network sources.

Finally, we evaluate the areas of the two *single-stream* RapidLink alternatives (i.e., *concentrated* and *lockstep-mode*) and the baseline wormhole network, when the NoCs are optimized to operate at 1 GHz. The total area occupied by an 8×8 NoC implemented using (a) baseline wormhole routers with 64-bit links, (b) a concentrated RapidLink organization, and (c) RapidLink routers operating in lockstep mode are: 0.87 mm^2, 0.73 mm^2, and 0.80 mm^2, respectively. For the two single-stream RapidLink configurations, the area numbers also include the area of the bridge modules at the network interfaces and the area of the DS-EBs placed on the links. The reported area analysis indicates that both RapidLink configurations require the least area, which is 16% smaller for the concentrated RapidLink, and 8% smaller for the lockstep-mode RapidLink architecture. If one also takes into account the high network performance of the concentrated RapidLink, one can safely conclude that it is the best architecture for single-stream networks.

4.4 Conclusions

The asymmetry between the NoC's intra- and inter-router delays has been exploited in many forms in the past, primarily aiming to allow flits to traverse longer distances within a single clock cycle. NoCs with high-radix routers constitute such an example; they allow flits to reach their destinations using fewer hops, albeit through the use of longer links and fairly complex routers. The increased number of ports complicates allocation and switching logic, and requires custom design to achieve acceptable operating frequencies. The design of high-radix networks typically leads to complicated layouts and wire-routing congestion, which also necessitate custom design effort. Additionally, high-radix NoCs incur higher latencies and power consumption when handling local traffic (e.g., near-neighbor), because of unnecessary data movement over longer distances.

Other alternatives employ single-cycle multi-hop link traversal by relying on complicated flow control rules and router bypassing to cross multiple hops “asynchronously” in one cycle. Once again, this philosophy does not offer a true benefit under localized traffic, which involves packets traversing one, or at most two, hops and requires significant redesign both at the micro-architectural and physical design levels.

In both above-mentioned philosophies, fast wire traversal is achieved assuming tile-based homogeneous systems that are characterized by a regularity in their physical layout. When longer links exist in the NoC, link pipelining should be adopted, which ruins the main property of delivering flits over a long distance in a single clock cycle.

On the other hand, the proposed RapidLink NoC architecture complements previous state-of-the-art proposals by following a distinct and more scalable design path, which improves network performance without increasing design cost. RapidLink is minimally intrusive to both the router's micro architecture and the flow-control policies, and it can be applied to any low- or medium-radix topology. RapidLink does not lose the benefits of local connectivity, and by using the proposed DDR dual-stream elastic buffers, one can split even longer links into multiple half-cycle DDR segments. RapidLink can be equally applied to single- or multiple-VC NoC configurations, while still offering significant network performance improvements and without increasing the hardware cost, as verified by extensive hardware implementation analysis using placed-and-routed designs.

Chapter 5

PhaseNoC: Traffic Isolation with TDM-Scheduled Virtual Channels

5.1 Introduction

Minimizing the NoC's hardware cost while not sacrificing network performance is a non-trivial objective, as the functionality expected from the NoC continues to grow. For instance, multi-core systems increasingly require some form(s) of isolation – or separation – among the traffic flows of concurrently executing applications. In the simplest case, such segregation attributes are desired due to restrictions imposed by higher-level protocols, e.g., cache coherence in CMPs. Constraints of this type are typically satisfied through static separation of flows (termed "message classes") using VCs within the NoC [26]. Nevertheless, the requirements of flow isolation are often more elaborate than mere physical partitioning, and include advanced rules that describe how the flows are allowed to interact, or the level of service that each flow should receive.

Flow *isolation* can manifest in two forms: (a) high-assurance, secure-grade *non-interference*, as required by systems demanding security guarantees [128], and (b) *performance* isolation with minimum-bandwidth and bounded-latency Quality-of-Service (QoS) guarantees [47]. Non-interference is becoming increasingly relevant within the context of virtual execution platforms, whereby the sharing of hardware resources must be impervious to cross-interference [47].

Existing solutions have only been able to satisfy one of the two above-mentioned objectives; i.e., either strict isolation at the cost of lower network throughput, or more efficient performance isolation without non-interference guarantees. There is currently no architecture that can selectively provide both properties, with minimal impact on overall network area and performance. The difficulty lies within the inherent nature of NoC architectures. Due to the widespread sharing of NoC resources among all the VCs, strict non-interference between flows is, by definition, not guaranteed. In fact, VCs are interfering by construction, since multiple VCs are eligible to compete for the same NoC resources at any given time.

The main goal of this work is to bridge the unseemly dichotomy between strict non-interference assurance and flexible/efficient QoS provisioning. Building on this fundamental premise, we propose the *PhaseNoC* architecture, which provides this dual flow-isolation property. The crux of PhaseNoC's operation revolves around the notions of *domains* and *phases*. A *domain* is defined as an individual VC – or a group of VCs – serving one (or more) application flow(s). Each PhaseNoC router can serve any number of domains in parallel, and it can guarantee strict isolation across the supported domains. The various domains are served in *phases* using a fresh re-interpretation of TDM, which is applied at the VC level. Overall, PhaseNoC is characterized by three fundamental properties, which highlight the key contributions of this work:

1. PhaseNoC relies on a complete overhaul of the routers' microarchitecture and their internal pipeline operation into a domain-amenable organization called *explicit pipelining*, which ensures that the various domains will never "compete" (i.e., arbitrate) with each other to gain access to any network resource. In fact, the domains are completely oblivious to each other's existence, and there is no information leak whatsoever across the domains, as highlighted in Section 5.2. Essentially, PhaseNoC constitutes a bandwidth-programmable traffic isolation mechanism that allows separate virtual networks – the supported domains – to operate concurrently and in complete isolation, despite sharing the same hardware infrastructure.

2. To address the latency overhead typically associated with TDM-based scheduling, PhaseNoC utilizes precisely choreographed phase propagation throughout the NoC, by applying VC-level TDM schedules at the granularity of individual router pipeline stages. The phases are coordinated into optimally scheduled interlocked propagating waves, which ensure that in-flight packets of all domains experience the minimum possible latency. A gen-

eralized methodology that enables the creation of optimal phase schedules in any network topology is provided in Section 5.3, which also includes weighted phase schedules, whereby bandwidth is unevenly distributed to the domains.

3. If secure-grade isolation is not required, PhaseNoC is also able to support a more "relaxed" traffic isolation mode that sacrifices the full non-interference properties to improve overall performance. PhaseNoC's *Opportunistic Bandwidth Stealing (OBS)* mechanism, introduced in Section 5.4, utilizes any bandwidth left unused by a domain in each cycle. The unused bandwidth is recycled among the domains that need it at any given time. The OBS technique cost-effectively facilitates the fusion of strict bandwidth guarantees and best-effort services, as encountered in mixed-criticality environments. Also, OBS allows for the seamless handling of bursty dynamic behavior, e.g., when a domain momentarily requires more bandwidth, beyond its statically allocated VC-level TDM quota. The extra traffic is "absorbed" by OBS, without affecting the other domains.

PhaseNoC is evaluated through extensive cycle-accurate simulations, as presented in Section 5.5, that include synthetic traffic patterns, and execution-driven full-system simulations with real application workloads. Comparisons against SurfNoC [128] – the current state-of-the-art in secure NoC architectures – and baseline NoC architectures that do not provide any isolation guarantees indicate that PhaseNoC provides consistently higher performance. A detailed hardware analysis using a 45 nm standard-cell library verify that PhaseNoC yields substantial cost savings, as compared to state-of-the-art VC-based architectures, despite providing the extra functionalities of both strict and relaxed flow isolation.

5.2 The PhaseNoC Router Architecture

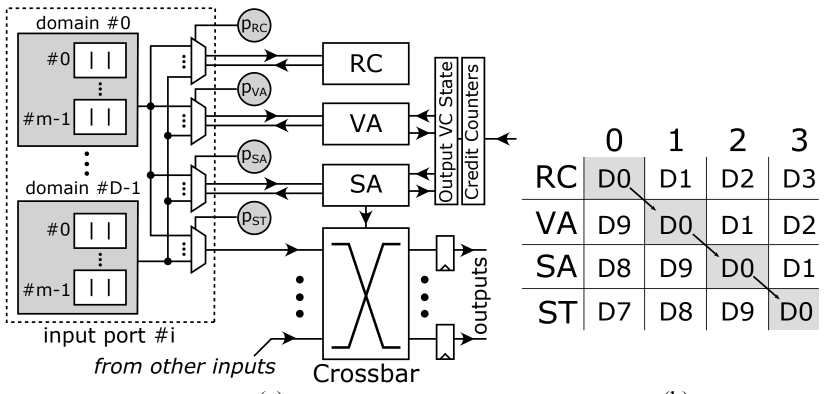
The PhaseNoC router follows an organization similar to the generic VC-based router architecture, presented in detail in Section 1.2.4. However, PhaseNoC organizes the allocation and the switching tasks executed per-packet and per-flit in phases, ensuring that each phase deals only with a distinct set of VCs (a so called *domain*). Every input port of the PhaseNoC router hosts V VCs that are organized in D domains, where each domain may contain a group of m VCs ($m = V/D$).

Non-interference is guaranteed if, at any allocation or switching step, the participating (competing) packets belong exclusively to the same domain (group of VCs). Thus, contention and interference can only arise between packets and flits of the same domain. PhaseNoC guarantees non-interference among all supported domains through its phased operation, i.e., each phase – covering all inputs of the router – deals exclusively with a single domain, and each phase is completely isolated (in terms of utilized resources) from other phases (and from other domains). The phase activation process should be the same for all inputs of the router, thus making it impossible for two different inputs to participate in a router's allocation/switching stage with packets/flits that belong to different domains.

5.2.1 Explicit Pipelining

When a router is pipelined, it may operate simultaneously on many application domains, by selecting the appropriate phase for each pipeline stage. Care should be taken to assure that two domains never simultaneously participate in the same stage. To ascertain this behavior, we let the router's pipeline operate in a predetermined (although programmable) static schedule. For example, once the group of VCs belonging to domain D0 perform VA, the group of VCs of domain D1 perform SA, while the winning flits of domain D2 pass the crossbar (ST), and the flits of domain D3 are on the link towards the next router. Therefore, in each cycle, the router is fully utilized, but each part of the router works on a different domain. This *unique* feature of PhaseNoC's pipeline ensures that each stage works on a different phase, and the flits/packets served in each phase belong exclusively to a single domain. We term this type of pipeline operation as *explicit pipelining*.

To achieve this behavior, each input port should own a separate path to VA, SA, and ST, as shown in Figure 5.1(a) (only the path to ST carries real data). Each input can send to each part of the router the requests/data of a different domain (group of VCs), provided that the select signals of the multiplexers (that coordinate the phase propagation) never point to the same domain. By setting the phase of each stage appropriately, all of them may be executed in parallel, but each stage acts on the packets/flits of a different domain. It is critical, however, that *all inputs* see the same order of phase activation. Additionally, secure-grade isolation can be achieved by building the multiplexers in front of each stage of the router using trusted logic [125], according to the guidelines in [128].



	0	1	2	3
RC	D0	D1	D2	D3
VA	D9	D0	D1	D2
SA	D8	D9	D0	D1
ST	D7	D8	D9	D0

(a) (b)

Figure 5.1: (a) The PhaseNoC router architecture. Only a set of VCs (one domain) is allowed to participate from each input in each allocation stage, and it is the same set (domain) for all inputs. One multiplexer for each stage, controlled by a TDM schedule, allows different domains to be served by different stages of the router. (b) Cycle-by-cycle operation of a 4-stage pipelined PhaseNoC router. In each cycle, all router parts are utilized, with each pipeline stage serving (allocating or switching) a different domain (group of VCs).

For zero-latency phase scheduling, a flit should always find the phase of its current pipeline stage aligned to its domain, and may move uninterrupted (unless it loses to another flit of the same domain during arbitration). For example, if the phase of VA in cycle 1 is serving domain 1, then the phase of the SA stage in cycle 2 should be serving domain 1 as well. This behavior is captured in Figure 5.1(b), which presents the cycle-by-cycle activity of an input port's pipeline stages. In each cycle (columns), all of the pipeline stages (rows) operate on a different domain, whose ID is shown by the number in the corresponding box. In the first cycle, RC operates in phase 0, so domain 0 is able to calculate its output port. In the next cycle, it finds its phase in VA, and it is able to allocate an output VC of its domain successfully, as flits of domain 1 perform RC. In cycle 2, domain 0 uses its allocated VC to participate in SA, while the head flits of domain 1 try to acquire an output VC. Whether this allocation is successful or not depends only on the contention appearing between the flits of domain 1 from all inputs; only domain 1 flits are allowed to participate in VA in this phase. In cycle 3, the flits from domain 0 that won in SA traverse the crossbar, and it is the turn of domain 2 to perform VA.

Explicit pipelining guarantees non-interference between flits/packets of different domains, irrespective of the packet length (flits per packet) and the pipeline depth of the routers. At any given time, each pipeline stage deals with the flits of a certain domain that may belong to packets of arbitrary length.

Routers with fewer pipeline stages can be built by merging stages. However, due to the phased operation of PhaseNoC routers, merging two stages means that their phase multiplexers should also be merged.

5.2.2 Structure of Allocators

In an N-port router with D domains and m VCs per domain, there exist a total of $N \times D \times m$ input and output VCs (recall that $D \times m = V$, i.e., the total number of VCs per input port of the router). The VA process between those input and output VCs in a traditional router would require an $N \times D \times m : N \times D \times m$ allocator. However, in PhaseNoC, in each clock cycle, only a single domain performs VA to a group of m VCs per input. Thus, in the whole router, at most $N \times m$ input VCs will try to allocate an output VC. Since an input VC will never request an output VC outside each domain, then at most $N \times m$ output VCs will be requested.

Thus, for completing the VA process in PhaseNoC, a simpler $N \times m : N \times m$ VC allocator suffices, which serves a different domain in each clock cycle. As illustrated in Figure 5.2(a), VA is performed in 2 stages. In VA1, each input VC of the domain matched to the current phase of the router selects one available and possibly ready (i.e., has at least one buffer slot) output VC. The selection is made by round-robin arbiters that select one of the m active VCs of an output port of the same domain. In VA2, each of the $N \times m$ output VCs is assigned through an $N \times m : 1$ arbiter to at most one input VC of the same domain.

Similar simplifications can be derived for the switch allocator as well, which, again, involves 2 steps of arbitration, as shown in Figure 5.2(b). The SA1 arbiter per input port is reduced from a $V : 1$ arbiter in a baseline implementation without domains to an $m : 1$ arbiter in PhaseNoC, since local arbitration involves only the input VCs of the domain currently active in the SA stage. The SA2 stage, which selects the input port that will access each output port, cannot be simplified further, and it still requires an $N : 1$ arbiter per-output.

Note that, if desired, the SA2 arbiters may be modified to implement any type of arbitration policy, and said policy would only affect the traffic *within* a particular domain. For example, the SA2 arbiters could implement policies such as round-robin, weighted arbitration [67], or

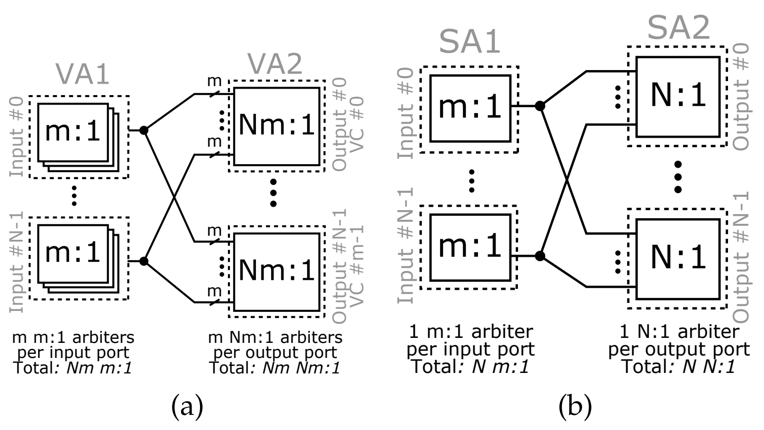


Figure 5.2: PhaseNoC's reduced allocators for the (a) VA, and (b) SA router pipeline stages.

even a static TDM schedule in the form of a TDM wheel [46]. The chosen arbitration policy will only affect the traffic *within* a domain, with no impact on the global TDM phase schedule. This functionality enables PhaseNoC to facilitate, if required, two-level schedules: the top-level TDM schedule coordinating the domain phases, and second-level schedules for the traffic within each domain.

Although the VA and SA allocators are shared by different domains, sharing is performed in time and, thus, it is impossible for packets that belong to two different domains to compete for the same resource in the same cycle. Additionally, to completely eliminate any domain interference, all arbiters should use D separate priority vectors, each one corresponding to the active domain. The appropriate set is selected by the phase of the allocation stage, ensuring that arbitration decisions are completely separated across domains.

5.3 Application-Domain Scheduling

PhaseNoC routers guarantee non-interference between different domains by time-multiplexing the allocators, the crossbar, and the output physical channels in different domains in each clock cycle. This time multiplexing scheme ensures that the latency and throughput of each domain is independent of the other domain's load; contention is only allowed between the packets of the same domain. To achieve zero latency overhead in flit propagation, the inter-router phase propagation should be extended to the network level to allow flits of one domain to travel in the network in a wave-like manner. The flits of each domain would traverse many hops in consecutive cycles, without waiting for the turn of their application domain to come. In this way, non-interference is achieved with the minimum possible latency, since only the flits of the same domain experience contention throughout the network.

5.3.1 Topology Constraints

Depending on the pipeline depth of the router, each router is concurrently active on one, or many, different application domains (groups of VCs). Therefore, once an application domain performs RC in cycle t_0, the same application domain will proceed to VA in cycle $t_0 + 1$, to SA in cycle $t_0 + 2$, to ST one cycle later, and it will eventually appear on the link (LT) in cycle $t_0 + 4$. Therefore, in order for the flits of this application domain not to experience any latency overhead, the router at the other side of the link should schedule the start of the service of this particular application domain in cycle $t_0 + 5$; the first step is again RC. So, for experiencing no latency between any two routers, the domain served in the first pipeline stage of both routers connected with a forward link should differ by $P + 1$ cycles; P, due to the router pipeline, plus one for the single cycle spent on the link.

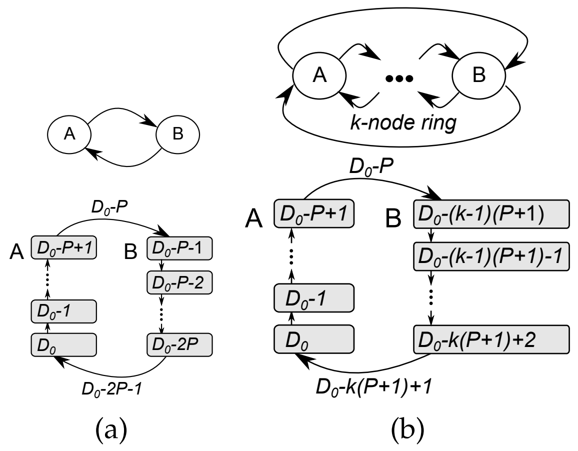


Figure 5.3: The scheduling constraints set by (a) direct, and (b) wrap-around links for achieving zero-latency flit traversal across routers.

To present the conditions that satisfy this property, we first study the simple case of two directly connected routers shown in Figure 5.3(a). We assume that the first pipeline stage of router A is serving domain D_0 in the current cycle. Under a zero-latency schedule, the first pipeline stage of router B will reach the domain currently served by the first pipeline

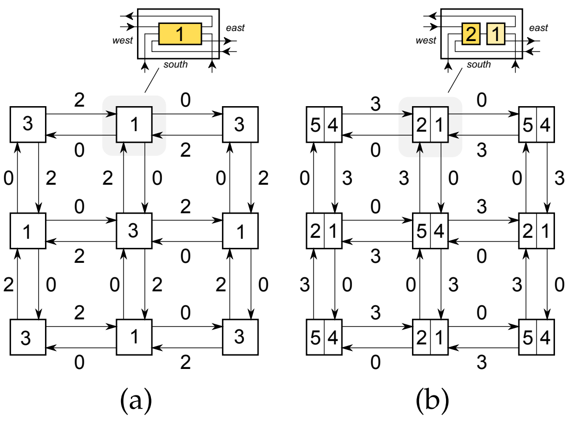


Figure 5.4: Zero-latency schedules on a 3×3 2D mesh using (a) 4 domains with single-cycle routers, and (b) 6 domains in 2-stage pipelined routers. All incoming links feed the first pipeline stage of the router, and all output links are driven by the last stage.

stage of router A after P cycles. At the same time, we should guarantee that this relationship between any two neighboring routers also holds in the backward direction, so that any traffic crossing router B towards router A does not experience any latency either. The output links of router B forward flits of domain $D_0 - 2P - 1$ when the first stage of router A is serving domain D_0. Therefore, if we want router A to receive in-sync the flits coming from B, then, in the next clock cycle, the next domain of A, which will be $D_0 + 1 \bmod D$, should match the domain that is served on the incoming links in this cycle, i.e., $D_0 - 2P - 1 \bmod D$, where D is the number of domains. This constraint is satisfied when

$$2(P+1) \bmod D = 0 \tag{5.1}$$

Condition (5.1) requires that the number of domains D is equal to any of the factors of 2, $P+1$, or $2(P+1)$.

Figure 5.4 depicts the assignment of 4 domains to PhaseNoC routers and, implicitly, to the network links, for a 3×3 mesh. Figure 5.4(a) shows a snapshot of the reset phase of the network, assuming single-cycle routers ($P = 1$, and, thus, $D = 4$), which select the same domain for all the inputs of the router. The number inside the nodes and next to the links corresponds to the domain ID. Then, each router independently increases its working domain by one in each clock cycle and wraps around when the number of domains is reached. Equivalently, Figure 5.4(b) shows the schedule applied to 2-stage pipelined routers (RC-VA in one cycle for one domain, and SA-ST in the next cycle for a different domain) that can serve 6 domains without any latency overhead.

Once the number of domains is selected appropriately, the assignment of starting phases to the internal pipeline stages of the routers and the links can begin from any node and propagate decrementally to the remaining nodes, while taking care to wrap around to domain $D-1$ when the domain 0 is reached. During the propagation of starting phases, all the incoming links of each router (and, as a result, all outgoing links, too) should serve the same application domain. This constraint is a requirement for offering isolation across domains, since on every pipeline stage inside the router the same domain is active on all inputs concurrently.

The proposed scheduling mechanism is extended to topologies that also contain wrap-around links, such as rings and tori. As depicted in Figure 5.3(b), two adjacent nodes may be connected with a forward and a backward link using a wrap-around connection in a k-node 1-D ring connection. In this case, and assuming that the first pipeline stage of router A is serving application domain D_0, the output links of router B (at the other end of the ring; k hops away) should be serving the domain $D_0 - k(P+1) + 1 \bmod D$, in order for any flits from B to A not to experience any latency. If a zero-latency penalty is also required for the flits that cross the wrap-around connection, then the domain currently being served by the outputs of router B should be equal to the domain that the first pipeline stage of router A will serve in the next cycle, i.e., $D_0 + 1 \bmod D$. Thus, for allowing a perfect schedule when wrap-around connections exist in the topology, we should guarantee that $(D_0 - k(P+1) + 1) \bmod D = (D_0 + 1) \bmod D$, which translates to:

$$k(P+1) \bmod D = 0 \tag{5.2}$$

The loop constraint of Equation (5.2) is satisfied when D is equal to any of the factors of k, $P+1$, or $k(P+1)$.

Figure 5.5 depicts a snapshot of the reset state of phase propagation in two Manhattan-grid networks. In Figure 5.5(a), the topology consists of 4-node rings in each dimension, which limit the number of application domains that can be hosted under a perfect (zero-latency) schedule to 8 domains, when routers operate in a single cycle. On the contrary, in Figure 5.5(b), the topology includes two sets of loops: the wrap-around rings that include 6 nodes and allow the scheduling of 12 domains with single-cycle routers, and the internal loops that span 4 nodes and limit

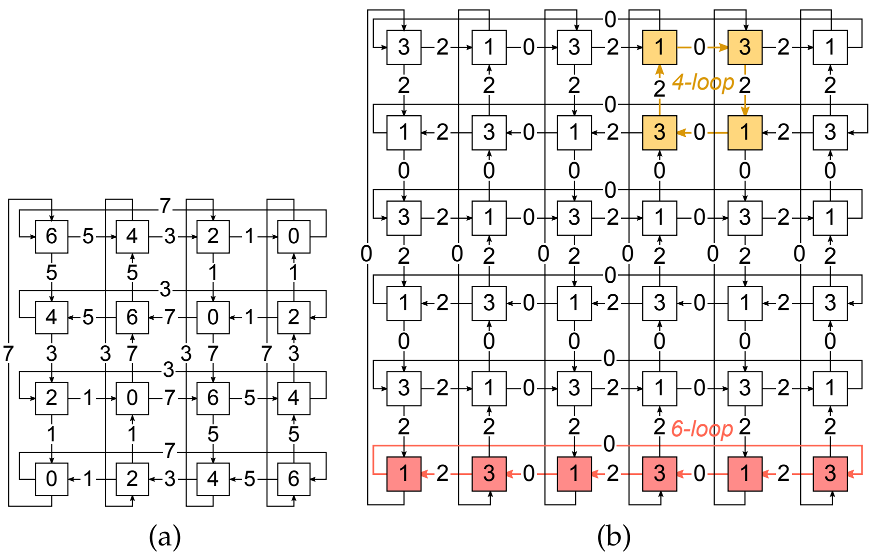


Figure 5.5: Examples of zero-latency schedules in two Manhattan-grid NoCs with single-cycle routers: (a) 8-domain schedules are allowed, constrained by the wrap-around rings; (b) only 4-domains can be supported here under perfect phase scheduling, due to the co-existence of two unequal loops in the NoC.

the supported domains to 8. However, since the two loops co-exist, the maximum number of domains that can be supported under a zero-latency schedule is $\gcd(12, 8) = 4$.

Similar limitations arise when wrap-around ring connections spanning k nodes (Figure 5.3(b)) co-exist with direct (self-loop) connections (Figure 5.3(a)), and, thus, the constraints of both connection patterns should be simultaneously satisfied. In this case, the number of domains D that can be scheduled with zero-latency overhead is equal to $\gcd(2(P+1), k(P+1))$. The number of domains is $2(P+1)$ or $P+1$ for even or odd values of k, respectively. For instance, as shown in Figure 5.6, a 6-node ring of single-cycle routers can host at most 4 application domains, i.e., $\gcd(2(1+1), 6(1+1))$. When k is odd and the topology includes both direct and ring loops, the number of domains can be increased from $P+1$ to $2(P+1)$, under a perfect schedule, by adding a pipeline register on one of the links of the k-node ring – thus delaying the flits for one more cycle and making the effective number of hops in the ring to become even.

The constraints derived by the direct (self-loop) connections between two neighboring routers and the wrap-around ring connections cover the majority of network topologies. However, in some cases, another

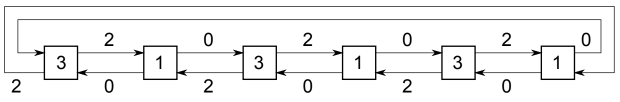


Figure 5.6: A zero-latency schedule of 4 domains in a 6-node ring network.

connectivity pattern exists, similar to the one shown in Figure 5.7. In this case, router A can reach router R following a multi-hop path through router B, or directly, using the express channel. When the first pipeline stage of router A serves domain D_0, then, at the same time, its output link is forwarding flits that belong to domain $D_0 - P$, which should be served by the first pipeline stage of B in the next cycle. Thus, node B must start by serving domain $D_0 - P - 1$ (similar to Figure 5.3(a)). In a similar manner, the first pipeline stage of router R, which is placed r nodes away, should be operating on domain $D_0 - rP - r$, in order for the flits of A (that reach R via B) to experience zero-latency overhead. However, A can also reach R directly. Therefore, in order for the two paths between A and R to be in-sync, and, thus, prohibiting any latency overhead, the equality $D_0 - P - 1 \bmod D = D_0 - rP - r \bmod D$ must hold. This equality is satisfied when

$$(r-1)(P+1) \bmod D = 0 \tag{5.3}$$

Equation (5.3) dictates that D should be equal to any of the factors of $(r-1)(P+1)$. If the condition does not hold, flits that arrive at the inputs of R from one of the two paths will have to wait for their phase to arrive.

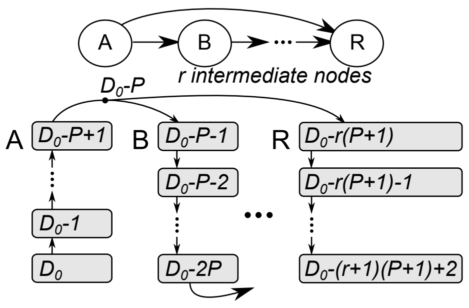


Figure 5.7: Zero-latency phase propagation on a re-convergent fan-out connection between routers A and R.

The conditions derived suffice to describe the connectivity patterns of *any network topology*, and how those patterns affect the number of application domains that can be supported by PhaseNoC, in order to allow flits to propagate without any latency overhead *irrespective of the path they take in the network*. The reason for this powerful attribute (i.e., no dependence on the path taken within the NoC) is due to PhaseNoC's operation, which ensures that all outgoing links of a router serve the same domain in a given cycle, following a perfect schedule. Thus, any adaptive routing strategy can freely select any output of the router and still enjoy in-phase flit propagation. In topologies, either regular or irregular, where the three connectivity patterns (or a subset of them) co-exist, the number of domains that can be supported under a perfect schedule is decided by the greatest common divisor of the number of domains given by conditions (5.1), (5.2), and (5.3). Once the number of domains is selected appropriately, and the phases are distributed on the links and the routers of the topology, a flit can flow uninterrupted from source to destination without experiencing any delay in any part of the network, and irrespective of the path that it follows, except: (a) the delay arising from contention with competing flits of the same domain, and (b) the time spent at the network interfaces while waiting for the turn of their application domain to come.

5.3.2 Extending the Number of Application Domains

PhaseNoC can support – by construction – an arbitrary number of domains; explicit pipelining does not limit in any way the number of active domains. However, at the network level, and depending on whether the number of domains are aligned to the identified topology constraints, the distribution of phases on the links and the routers of the NoC may lead to (a) either a perfect schedule with zero-delay overhead, or (b) it will inevitably favor some paths relative to others, in terms of delay. When we cannot align the number of domains to match the conditions required for perfect scheduling, we can follow another approach, which enables the scheduling of a larger number of domains with a controllable latency overhead.

More domains can be supported after topology partitioning, which aims to remove one or more of the topology constraints. For example, the tree topology shown in Figure 5.8, which consists of single-cycle routers, can host at most 4 domains due to the self-loop constraint in each bidirectional link (condition (5.1)). The self-loop constraint can be broken if we separate in different sub-networks the up and down connections. In this way, the number of domains that can be supported under a perfect schedule by PhaseNoC is unlimited in each sub-network, since there

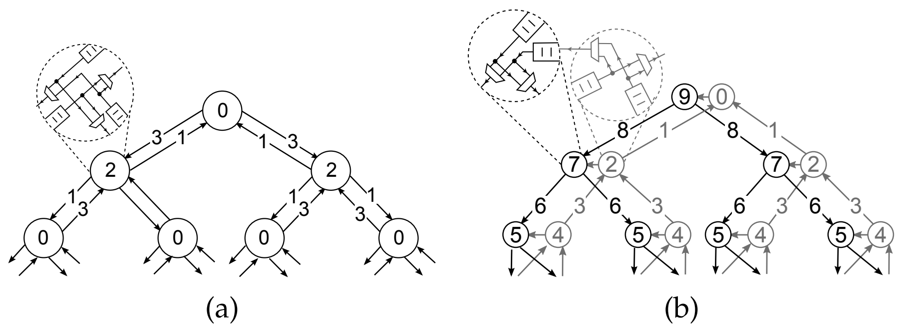


Figure 5.8: Phase propagation of (a) 4 domains in a tree topology, and (b) 10 domains in a partitioned tree topology (up and down connections separated).

is no loop that would limit the number of supported domains under a perfect schedule. Figure 5.8(b) examines the case of 10 domains.

In other cases, even though partitioning significantly increases the number of domains that can supported under a perfect schedule, said number is not unlimited. For example, the hierarchical ring built from single-cycle routers shown in Figure 5.9(a) can host at most 2 domains (the greatest common divisor of the number of domains due to the self-loop, the 3-node loop of the local rings, and the 6-node loop of the global ring). After partitioning the rings, by separating the left and right directions, the constraints are more relaxed and 6 domains can be hosted, as shown in Figure 5.9(b).

Perfect scheduling holds inside each sub-network and does not extend across sub-networks. Once a flit finds its domain active, it will move uninterrupted until it has to change direction and turn into the other sub-network. In order for the flit to turn, the appropriate phase should arrive. Assuming, in the general case, that the same D-slot schedule is running in both sub-networks, the flit may need to wait at most D cycles for its domain to arrive, since the schedules in each sub-network are independent. However, this will happen at most once in the flit's path; whenever the flit moves from one sub-network to the other.

This strategy can be followed in any topology, thus extremely simplifying the perfect scheduling requirements. For example, a 2D mesh can be split in two sub-networks, one traversed by flits moving in the X+/Y+ directions, and the other one by flits moving in X-/Y-. In that case, flits would experience latency only when turning from X+ to Y-, or X- to Y+, assuming XY routing. A partitioned 2D mesh is the only topology applicable to SurfNoC's schedules [128]. On the contrary, in this work, we

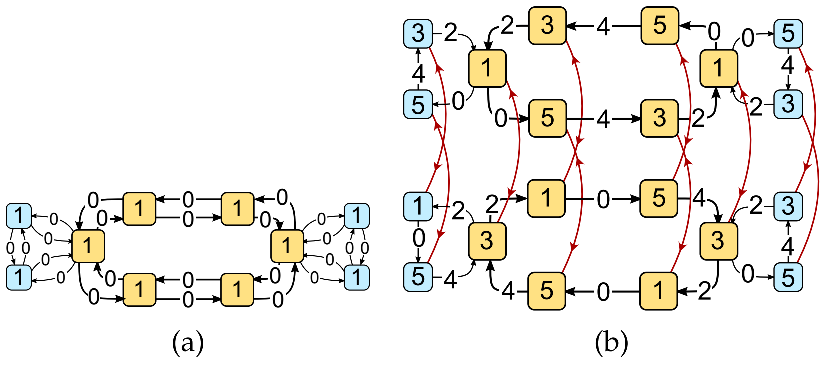


Figure 5.9: Phase propagation of (a) 2 domains in a hierarchical ring topology, and (b) 6 domains in each sub-network of a partitioned version of the same topology (i.e., the hierarchical ring topology of (a) is partitioned, by separating the left and right directions). Phases across sub-networks are not aligned.

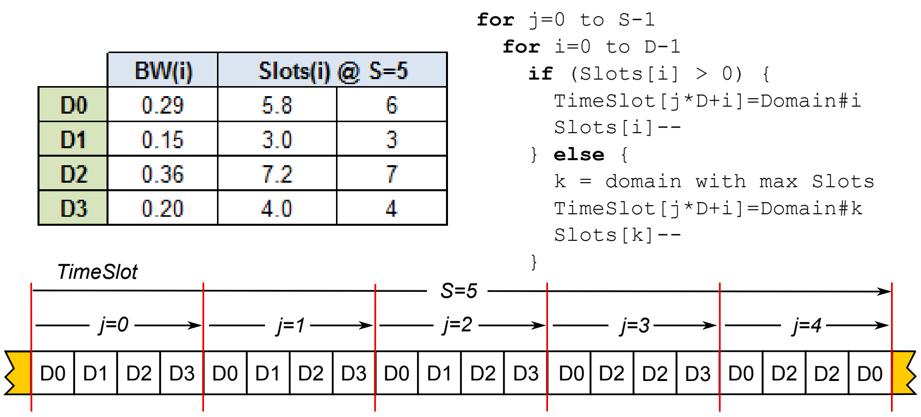
have identified the constraints for zero-latency phase propagation in *arbitrary topologies*, showing also how the aforementioned constraints can be relaxed by selectively breaking the loops of the topology. Finally, although TDM NoCs that apply TDM-based contention-free routing (per-output SA2 arbiters are replaced by TDM wheels) – such as [46, 55, 122] – do not face any topology restrictions, the actual topology constraints appear implicitly as design-time scheduling inefficiencies that lead to low TDM slot usage.

5.3.3 Weighted VC-level TDM Schedules

The bandwidth of the network may be unevenly distributed across application domains – in a programmable manner – by just changing the TDM schedule of phase activation in each PhaseNoC router. When each domain receives an equal share of the network's bandwidth, its phase is activated once every D cycles (the period of the TDM schedule is exactly D cycles), while, at the same time, it enjoys zero-latency traversal, as facilitated by the phase propagation in the network. When the bandwidth is not equally shared across domains, TDM scheduling expands to more than D cycles.

One time-frame of bandwidth allocation (i.e., VC-level time slots) consists of S TDM sub-periods of D cycles each. The minimum bandwidth that can be allocated is 1 slot every $S \times D$ cycles. The value of S determines the granularity of bandwidth allocation. The minimum value of S is equal to $S_{\text{min}} = \left\lceil \frac{1}{B_{\text{min}} D} \right\rceil$, where B_{min} represents the smallest bandwidth given to any of the application domains, and it is normalized to the maximum bandwidth of 1 (i.e., 100%). On the other hand, the maximum value of S can be arbitrarily large to enable very fine-grained bandwidth allocation. For practical purposes, we can compute S assuming that $B_{\min}$ is the smallest non-zero difference between the bandwidth allocations to the various domains. This ensures sufficiently fine granularity in the allocation.

For example, assume the case of 4 application domains ($D = 4$), which are programmed to receive 0.29, 0.15, 0.36, and 0.20 of the total bandwidth, as illustrated in Figure 5.10. In this case, $B_{\min} = 0.15$ (the minimum allocated bandwidth quota), so the value of $S_{\min} = 2$ can be used. The corresponding application domain should receive 1 time slot every 8 cycles, which is less than the bandwidth required. Also, by using only 2 TDM periods, we will not be able to approximate closely the rest user-defined bandwidth ratios. On the contrary, and as previously mentioned, we can use the minimum non-zero bandwidth difference across the domains to improve the granularity of bandwidth allocation. This minimum difference is equal to 0.20-0.15=0.05, which requires $S = \left\lceil \frac{1}{0.05 \times 4} \right\rceil = 5$ TDM periods to achieve finer granularity. In this case, $S \times D = 20$ time slots must be periodically allocated to 4 application domains (see Figure 5.10).



	BW(i)	Slots(i) @ S=5	
D0	0.29	5.8	6
D1	0.15	3.0	3
D2	0.36	7.2	7
D3	0.20	4.0	4

```
for j=0 to S-1
  for i=0 to D-1
    if (Slots[i] > 0) {
      TimeSlot[j*D+i]=Domain#i
      Slots[i]--
    } else {
      k = domain with max Slots
      TimeSlot[j*D+i]=Domain#k
      Slots[k]--
    }
```

Figure 5.10: Example of the construction of a weighted TDM phase schedule, which enables uneven allocation of bandwidth to the supported domains.

Each domain should receive $BW(i) \times S \times D$ VC-level time slots, where $BW(i)$ represents the user-defined bandwidth share of the ith domain. In the example of Figure 5.10, domains should receive 5.8, 3, 7.2, and 4 time slots, according to their required bandwidth share. When the number of VC-level time slots is not an integer, then – inevitably – one domain will receive one additional time slot, while another domain will receive one less time slot. Many rules can be applied to guide the distribution of the left-over time slots to the various application domains. In our example (Figure 5.10), we select to round the time slots to the closest integer. Consequently, domain D0 will receive 0.30 of the bandwidth (instead of the requested 0.29), while domain D3 will receive 0.35 (rather than the requested 0.36). If we want the actual bandwidth allocation to approximate more closely the requested bandwidth allocation, we must increase the value of S (thereby increasing the granularity of bandwidth allocation).

The phase activation of the application domains should be programmed to be repeated every $S \times D$ cycles, while still respecting the following two rules of PhaseNoC scheduling: (a) PhaseNoC guarantees that the domains allocated at least $1/D$ of the bandwidth will experience a perfect schedule for this portion of their allocated bandwidth. The additional VC-level time slots given to any domain (on top of $1/D$) will be out-of-phase, but without any latency penalty, since the domain will be activated more than once every D cycles in some TDM periods. (b) The domains receiving a bandwidth less than $1/D$ will operate partially in-phase, since their domain will not be activated in all TDM periods; thus, more than D cycles may elapse between two consecutive activations of the low-bandwidth domain.

The allocation algorithm, illustrated in Figure 5.10, maps the predetermined number of time slots (Slots[i] for the ith application domain) to the $S \times D$ time slots of the bandwidth allocation time-frame. As long as a domain has not consumed all of its available time slots, it receives a time slot that is either fully, or partially, in-phase. A domain is always scheduled in-phase, as long as it requires one time slot in all TDM sub-periods (i.e., its requested bandwidth is at least $1/D$). When a time slot is empty – since it belongs to a low-bandwidth domain that has used all of its slots – the slot is assigned to an out-of-phase domain (but with shorter delay) that has the maximum number of remaining slots.

5.4 Boosting Performance with VA Concurrency and Opportunistic Bandwidth Stealing

PhaseNoC's scheduling approach leads to a static (albeit programmable) allocation of bandwidth to domains. Using the VC-level TDM scheduling mechanism, PhaseNoC guarantees that packets injected in one application domain cannot affect the delivery of packets in a different application domain. More specifically, due to the strict schedule, each domain will get a static share of the network's bandwidth, irrespective of the traffic flowing in other domains. Even if a "rogue" flow tries to flood the network, all other packets belonging to different domains will not be affected at all, both in terms of throughput and latency. Each domain is completely impervious to interference, and PhaseNoC's secure-grade isolation ensures that no information leaks across domains, i.e., a domain cannot infer anything pertaining to any other domain's traffic intensity. The downside of these strict guarantees that is typical for any architecture that offers strict network traffic isolation, lies in the fact that a domain cannot enjoy *more* bandwidth than the portion pre-allocated to the domain in the static schedule. This is true even if some domains use fewer of their available time slots at any given time.

When strict isolation guarantees are not necessary, it would be beneficial to reclaim unused cycles to decrease latency and improve throughput. Moreover, by facilitating some form of bandwidth "stealing" across domains, the NoC would be able to accommodate sudden traffic bursts and maintain work-conserving transmission of data. Toward this end, we present two light-weight modifications to the PhaseNoC microarchitecture that work synergistically (a) to minimize unwarranted latency overheads, and (b) to "recycle" any bandwidth left unused by the domains. The latter feature may sacrifice the *secure-grade* non-interference guarantees, but it still provides *complete performance isolation* to each domain. In other words, a domain may now be able to deduce the traffic intensity levels in other domains (i.e., information leak is now possible), but each domain is still provided with *at least* its pre-allocated performance guarantees, irrespective of the traffic intensity in other domains. Depending on the system's characteristics, PhaseNoC can be configured to support either strict non-interference, or performance-only isolation.

5.4.1 Phaseless VC Allocation

The first modification targets the latency overhead incurred due to the per-packet operations. For instance, the VA pipeline stage concerns only the *head* flit of each packet. Thus, if we provide time slots for VA to all flits in the network, said time slots will be left unused. This becomes especially pronounced in pipeline setups where VA is executed in different pipeline stages from SA and the non-head flits are coerced to traverse the VA stage without doing any actual work.

Therefore, a *phase-less* VA would be preferable, which would not consume any valuable slots in the TDM schedule, and which would be executed only by head flits. In order to enable such optimization and achieve maximum utilization, *packets from multiple domains must be allowed to concurrently execute VA*, while still guaranteeing that the VA operation of any packet does not impact any other packet in a different domain. In a baseline VA unit, interference among input VCs can only occur in the VA2 stage, if multiple input VCs request the same output VC. Note that no interference occurs in VA1, since the latter is simply a selection process executed independently by each input VC. If each output VC is only allowed to be requested by input VCs belonging to a single domain, then no two domains can interfere (arbitrate against each other) in VA2. Therefore, input VCs from different domains are prohibited to request the same output VC. This restriction constitutes a sufficient condition to allow input VCs from multiple domains to perform VA in parallel, without any cross-domain interference.

To enable concurrent execution of VA by multiple domains in each cycle, the simplifications to the VA allocators outlined in Section 5.2 must be relinquished in favor of a baseline VA unit that would allow more head flits per cycle to allocate an available output VC of the same application domain.

5.4.2 Opportunistic Bandwidth Stealing

The second modification to the baseline PhaseNoC architecture is the augmentation of a mechanism called *Opportunistic Bandwidth Stealing (OBS)*. This technique optimizes the SA stage by aiming to maintain all output ports of the router busy in each cycle, whenever the active domain would otherwise leave some output ports unutilized. The OBS mechanism observes which output ports are left unused in each cycle by the active domain. All head-of-line flits belonging to the currently active domain are known as *in-phase* flits. The unused output ports in each cycle are thereby used by *out-of-phase* flits, i.e., flits belonging to other (inactive) domains. Hence, any bandwidth left unused by in-phase flits in each clock cycle is opportunistically "stolen" by out-of-phase flits. To achieve this without violating performance isolation properties, the new policy must comply with the following two-fold rule, which strictly prioritizes in-phase flits over out-of-phase ones: an out-of-phase flit is allowed to participate in SA if (a) there is no in-phase flit eligible to

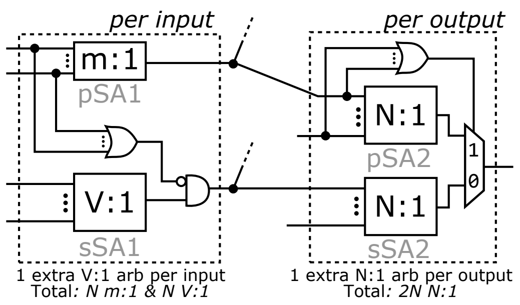


Figure 5.11: The modified SA architecture employed by PhaseNoC's OBS mechanism to enable the opportunistic stealing of idle (unused) output port slots by out-of-phase flits.

participate in SA in the out-of-phase flit's input port, and (b) there is no in-phase winner heading to the out-of-phase flit's destined output port.

In order to implement this double-faceted rule with a reasonable implementation overhead, we introduce a *secondary SA* unit (sSA), which handles switch requests from out-of-phase flits independently. The sSA unit operates in parallel with the *primary SA* unit (pSA), which still operates exactly as described in Section 5.2 and serves requests from flits of the active domain (i.e., in-phase flits).

Similar to a separable input-first SA implementation, the sSA module consists of two stages of arbitration, as illustrated in Figure 5.11. At the input side, sSA1 serves the requests from VCs that do not belong to the active domain, but have already been allocated an output VC (with at least one free buffer slot). The winner of sSA1 is allowed to proceed to the second stage, only if pSA1 has not declared any in-phase winner for that input port. As shown in Figure 5.11, the grants of sSA1 are masked, when at least one request was made to the local pSA1 unit. Output port requests from the winners of sSA1 reach the second arbitration stage, sSA2, which declares an out-of-phase input port winner. However, the final winner for every output port is decided after checking the outcome of the corresponding pSA2 unit. If pSA2 has produced a grant, an in-phase flit is eligible to access the specific output port, and it is given priority through the multiplexer on the right-hand side of Figure 5.11. In the absence of an in-phase winner, the output port is used by the out-of-phase sSA2 winner.

With OBS's secondary SA policy, all domains are still guaranteed their assigned TDM schedule slots. However, if these slots are unused in any given cycle, another domain can opportunistically steal them. Essentially, OBS complements strict provisioning of bandwidth guarantees with best-effort services, in a very efficient and cost-effective manner that allows seamless handling of bursty traffic; when a flow momentarily exceeds its statically allocated bandwidth quota (e.g., due to dynamics in an application's behavior), OBS can absorb the excess traffic without affecting the other domains.

By allowing *all* out-of-phase flits to always compete for the unused bandwidth of *any* in-phase domain, OBS inevitably sacrifices the strict two-way isolation of PhaseNoC (information leak can happen across any two domains). However, OBS could easily be modified to forbid out-of-phase flits of particular domains from "stealing" unused bandwidth from particular in-phase domains, by selectively masking certain out-of-phase requests. Such controllability would enable configurable and directional (one-way) isolation under OBS, if desired.

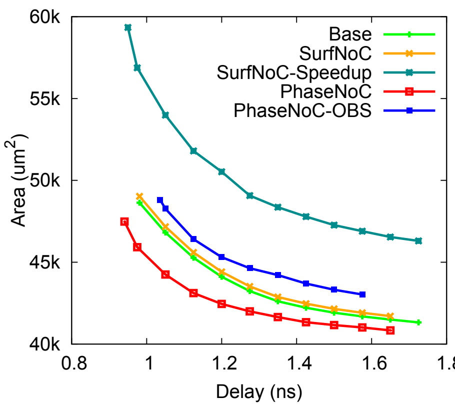
5.5 Experimental Evaluation

5.5.1 Hardware Implementation

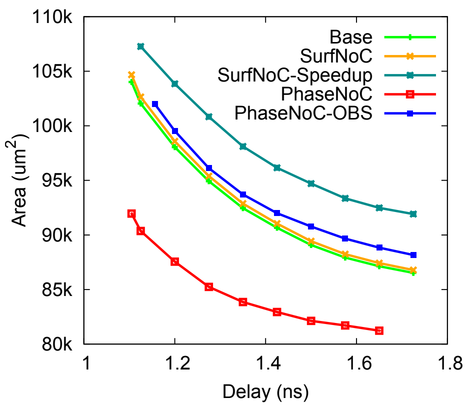
The first step in the evaluation of PhaseNoC is the quantitative assessment of the hardware complexity of the proposed designs relative to the state-of-the-art. PhaseNoC can be applied to any topology and any flow-control mechanism. The only part of the NoC that is affected by PhaseNoC is the router architecture, which now operates under explicit pipelining. Therefore, any comparison at the router level directly reflects the overall benefits, or possible overheads, of the PhaseNoC concept at the NoC level.

Five different 2-stage pipelined architectures are considered: (1) a baseline NoC design with no traffic-isolation, or QoS-provisioning capabilities; (2) the SurfNoC architecture [128] with no input speedup, (3) SurfNoC with input speedup equal to the number of supported domains (SurfNoC-S); (4) the proposed PhaseNoC design providing, *secure-grade* domain isolation; and (5) the PhaseNoC architecture augmented with the OBS mechanism (PhaseNoC-OBS), which optimizes network performance while still guaranteeing complete *performance* isolation. The SurfNoC-S design follows the original architecture proposed in [128], which requires crossbar input speedup to achieve acceptable performance.

Our goal is to evaluate the benefits arising from the explicit operation of the router in phases, which limits the allocation and switching oper-



(a) 4 VCs in 4 domains



(b) 8 VCs in 4 domains

Figure 5.12: Area-delay curves for 2-stage pipelined routers of baseline, SurfNoC, SurfNoC with input speedup, PhaseNoC, and PhaseNoC-OBS NoC architectures. Two configurations are investigated: (a) 4 VCs, and (b) 8 VCs organized in 4 domains.

ations to the input/output VCs of a certain domain. Figure 5.12 depicts the area-delay curves for the 2-stage pipelined routers under comparison. All routers are tested under two different configurations, shown in Figures 5.12(a) and 5.12(b), respectively: (a) 4 VCs with 1 VC per domain and (b) 8 VCs organized as 2 VCs per domain. For the baseline routers, we assume that the 4 and 8 VCs are grouped in 4 virtual networks. All routers have 4 buffers per VC as needed to cover the credits' RTT in 2-stage pipelined routers.

At every design point, PhaseNoC routers require the lowest area, while being as fast as the fastest state-of-the art design. More specifically, PhaseNoC occupies up to 20% smaller area than the state-of-the-art SurfNoC design – which employs input speedup – without any delay overhead. Moreover, PhaseNoC requires 13% less area and achieves 5% lower delay than a baseline NoC. The reported savings are mostly a result of the complexity reduction in the allocation units. PhaseNoC's allocation is always limited to the input VCs of one domain, which allows both sharing of the arbiters, and a reduction in their logic depth. Increasing the number of domains will increase the savings accordingly.

PhaseNoC-OBS adds a minimal (less than 6%) overhead in area and delay, as compared to the baseline, while still being more area efficient than SurfNoC with input speedup. PhaseNoC-OBS uses the same VC allocation logic as baseline designs, while it employs a secondary switch allocator that operates in parallel to the main one (to opportunistically "steal" the unused time slots of the VC-level TDM schedule). SurfNoC without input speedup behaves almost identically to the baseline. However, it offers the worst network performance, as will be shown in the following sub-section.

5.5.2 Network Performance Evaluation

The goal of this sub-section is to evaluate the effectiveness of PhaseNoC, in terms of network performance, as compared to conventional VC-based NoCs and SurfNoC architecture [128]. Moreover, the traffic isolation properties of the various architectures are also investigated.

In the first set of experiments, we simulate a 64-core tiled CMP system running real application workloads on a Linux operating system, configured according to Table 1.1. We assume that the system operates on two domains. The first domain executes a multi-threaded application from PARSEC benchmarks [20], while the other domain receives synthetic GPU traffic. Three VCs are assigned to the domain serving the PARSEC application, and three other VCs to the domain serving the synthetic GPU traffic. This VC setup was dictated by the MOESI cache-coherence protocol, which requires at least three virtual networks to prevent protocol-level deadlocks. Each VC has a depth of 3 flits, which is adequate to cover the credits' RTT in single-cycle routers. The synthetic GPU traffic is modeled based on real GPU application behavior, as described in [14], and exhibits the *many-to-few-to-many* traffic patterns observed in GPU applications.

The five architectures under comparison are configured to support two domains: one for the PARSEC applications, and one for the synthetic GPU traffic. Each domain receives 50% of the offered bandwidth. The injection rate of the synthetic GPU traffic is 0.054 flits/node/cycle, which is, in fact, rather high, because the many-to-few-to-many traffic patterns create severe hot spots in the network [14].

Figure 5.13(a) shows the *average network latency* of the various PARSEC applications, normalized to the baseline NoC. Recall that GPU traffic also traverses the NoC, concurrently with the PARSEC application traffic. As expected, the baseline router provides the best performance, albeit with no isolation guarantees. SurfNoC provides complete isolation (non-interference) to the PARSEC applications, but at a significant performance cost (45% latency increase, on average). When SurfNoC employs input speedup, this cost decreases to 19% on average, but with a substantial area overhead, as shown in Section 5.5.1.

On the contrary, PhaseNoC's performance is substantially better than SurfNoC and close to SurfNoC with input speedup, while still providing complete non-interference. This performance improvement is attributed to the efficiency of PhaseNoC's zero-overhead phase scheduling. On average, PhaseNoC's latency overhead over a baseline NoC is 16%. Finally, PhaseNoC-OBS, which sacrifices secure-grade isolation for performance-only isolation, provides near-identical performance as the baseline NoC.

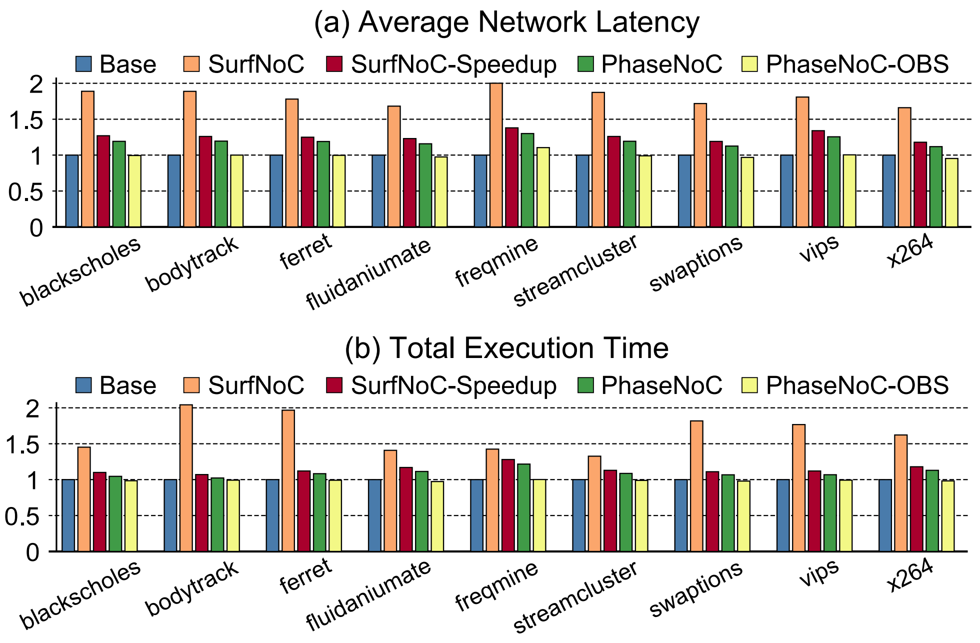
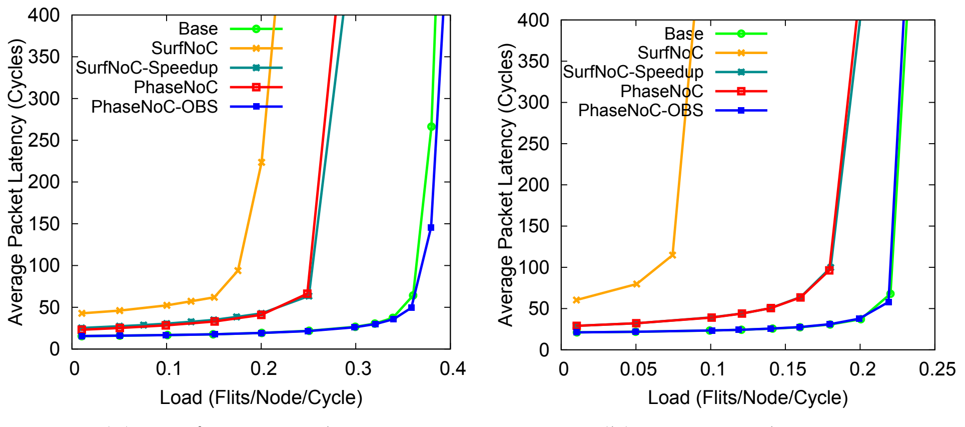


Figure 5.13: Full-System simulation results using PARSEC applications [20].

Similarly, Figure 5.13(b) shows the *total execution time* of the various PARSEC applications, normalized to the baseline NoC router. The trends are similar to those of the average network latency, but PhaseNoC is now only 8% slower, on average, than the baseline NoC. This indicates that the relatively small degradation in average network latency sustained under PhaseNoC does not greatly impact the total execution time (due to the fact that a lot of the packets exhibit large slack [32]). Once again, PhaseNoC-OBS offers near-identical performance as the baseline NoC (the differences are within simulation-noise margins). Overall, both PhaseNoC incarnations yield execution times that are very close to the baseline, with significantly less area than baseline and SurfNoC designs.

Despite the authenticity provided by full-system simulations, the flexibility to stress the NoC and isolate its inherent attributes is somewhat limited, due to the fixed characteristics of the running applications. In order to differentiate more clearly the various architectures, we henceforth resort to *synthetic* traffic patterns (for the remaining experiments). Various configurations are explored, while the network size, routing algorithm, and buffer depth are the same as in the full-system simulation configuration. For the latency-throughput analysis, we assume a bimodal distribution of packets with 50% of the packets being short, 1-flit request packets, and the rest being long, 5-flit reply packets.

In the first set of experiments, we examine the average network performance characteristics of the architectures under comparison. Figure 5.14(a) depicts the load-latency curves of PhaseNoC architectures, baseline NoC (Base), and SurfNoC variants, assuming single-cycle routers with 4 VCs per port, which are distributed to 4 domains (one VC per domain), under Uniform Random (UR) traffic. The baseline NoC, even if not providing any isolation properties, assumes that each VC is a virtual network, thus in-flight VC changes are prohibited. The PhaseNoC and SurfNoC variants are driven by a static 4-phase TDM schedule that gives an equal share of the available bandwidth to each domain. The same setup holds in Figure 5.14(b), which repeats the same experiment using the Bit Complement permutation traffic pattern.



(a) Uniform Random (b) Bit Complement

Figure 5.14: Average packet latency as a function of the total load for (a) Uniform Random, and (b) Bit complement permutation traffic.

From both figures, we can deduce that the zero- or low-load-latency of PhaseNoC, even after the strict TDM operation at the VC level, is still very close to that of the baseline design, due to the efficient network-level schedule, which allows the domino-like propagation of flits without any additional latency (independent of the source and destination, the topology, or the routing algorithm). In terms of saturation throughput, PhaseNoC offers lower throughput than the baseline, due to the non-interfering operation across domains, which reduces – in some sense – the choices for flit interleaving. Note that this difference is not observed in the real applications of Figure 5.13, because the injection rates of said applications are very low. This throughput loss is inevitable, since there is no way for the time slots allocated to one domain to be used by another domain. On the contrary, PhaseNoC-OBS manages to fully recycle all unused time slots. SurfNoC *without* input speedup provides the worst performance. The wave propagation in SurfNoC adds additional latency when moving from a south-east direction to a north-west one. Further, the absence of crossbar input speedup markedly increases SurfNoC's latency overhead. On the contrary, SurfNoC *with* input speedup improves the performance dramatically, thereby yielding results that are near-identical to PhaseNoC. However, this performance comes at a steep hardware cost: an 20% area overhead relative to PhaseNoC. The PhaseNoC-OBS design substantially outperforms both SurfNoC variants and achieves performance that is on par with the baseline NoC.

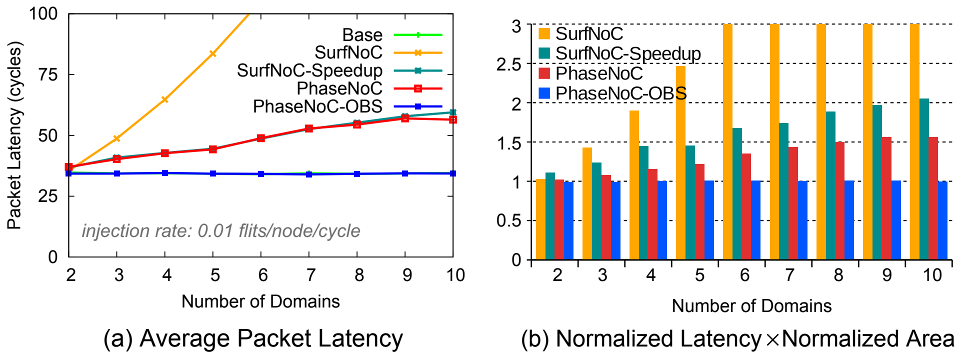


Figure 5.15: Comparison of PhaseNoC and SurfNoC networks using 4-stage pipelined routers when varying the number of application domains, in terms of (a) average packet latency, and (b) Normalized Latency × Normalized Area (the normalization is to the baseline NoC).

To evaluate the effect of the number of application domains on latency and hardware area, we conducted an additional experiment. Figure 5.15(a) shows the average packet latency of PhaseNoC and SurfNoC networks using 4-stage pipelined routers at an injection rate of 0.01 flits/node/cycle using UR traffic, when varying the number of application domains. In terms of latency, PhaseNoC and SurfNoC with speedup show equivalent performance, both experiencing a controlled latency overhead. On the other hand, PhaseNoC-OBS can “absorb” any extra delay due to phase non-alignment, thus reducing the criticality of scheduling and phase alignment. Figure 5.15(a) does not capture the impact of supporting additional domains on *hardware cost*, which is very high for SurfNoC with input speedup (as previously analyzed). To capture the interaction between latency and area, we employ the combined metric of $(Latency_{design}/Latency_{base}) \times (Area_{design}/Area_{base})$. Under this metric (lower is better), both PhaseNoC variants exhibit the best performance, as shown in Figure 5.15(b). Specifically, PhaseNoC outperforms SurfNoC *with* input speedup by 18%, in terms of this combined metric.

Note that the support of additional domains implies additional area cost, due to the presence of additional VC buffers. Hence, PhaseNoC’s area efficiency over SurfNoC with input speedup (20% lower area) allows for the support of *additional* domains. Conversely, assuming equal router-area budget, PhaseNoC can support more domains than SurfNoC.

PhaseNoC variants achieve high network performance, while still offering isolation properties that are not provided by a baseline NoC. The first scenario that shows the isolation properties of PhaseNoC, involves a 2-domain network, where each domain has 1 VC, in both PhaseNoC and a baseline NoC. The traffic on both VCs is uniform-random, while the load in VC0 remains constant and the load in VC1 is progressively increased. As shown in Figure 5.16, the load in VC0 of PhaseNoC is completely immune to the changes in VC1’s load. However, in the baseline case, as the load in VC1 increases, the accepted throughput of VC0 (domain 0) drops and converges to that of VC1 (domain 1).

In the second scenario, we evaluate the bandwidth guarantees offered by PhaseNoC when enhanced with the OBS mechanism, as compared to a baseline network without any guaranteed services. We simulate a network with 2 domains, each one utilizing a single VC, which are used to serve two different traffic classes: Best-Effort (BE), and an urgent Guaranteed-Bandwidth (GB) service. To provide such services, PhaseNoC with OBS operates under a schedule that offers 99% of the slots to the GB domain, leaving only the remaining 1% to BE. In this way, GB traffic is always guaranteed high priority, while BE can still enjoy more than 1% (as long as GB is idle), in order to deliver the best-effort service. In this experiment, BE is serving a constant load of uniform-random traffic, while GB remains idle most of the time and is only used

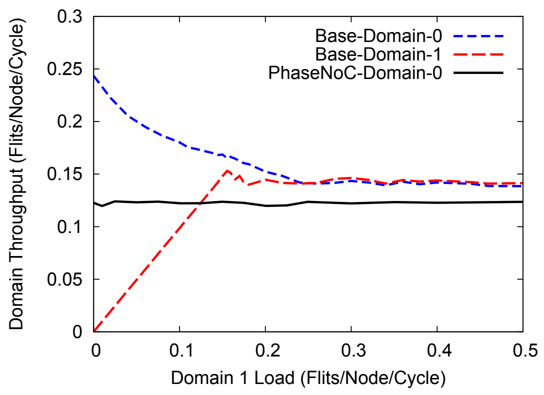
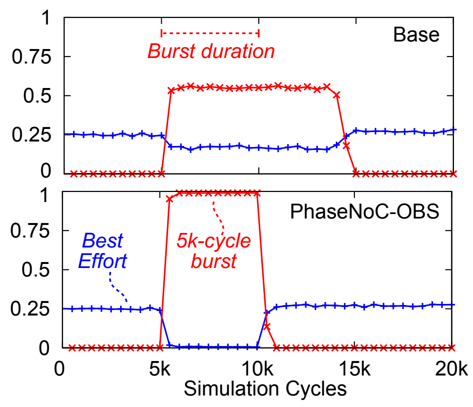


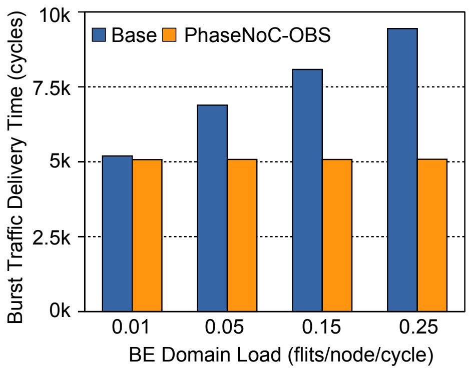
Figure 5.16: Non-interfering isolation of traffic in PhaseNoC. Unlike a baseline NoC, traffic in VC0 (i.e. Domain 0) of PhaseNoC is constant, irrespective of the load behavior in the other domains.

to inject urgent burst traffic, once during the simulation time: On the 5K-th cycle, 12 random nodes initiate a 5K-cycle burst traffic, each one towards a separate destination node, using the GB domain.

Figure 5.17(a) illustrates both domains' accepted throughput during the simulation period, whereby the BE load is set to 0.25 for both the baseline (Base) and PhaseNoC-OBS networks. Before the bursts are initiated, the BE traffic in PhaseNoC-OBS "steals" the idle domain's slots, and is able to deliver the same throughput as the baseline network. Once the burst injection starts, the urgent traffic (GB) is offered higher priority and immediately receives 99% throughput, in contrast to the baseline network, where the traffic is equally shared among the two domains, without any prioritization. Immediately after the burst stops, PhaseNoC-OBS has delivered the urgent traffic in time, and the BE domain immediately recovers. On the other hand, the recovery period in the baseline network is almost as long as the duration of the burst, since urgent traffic has not been ejected yet, and occupies network resources long after the burst is over. The total delivery time of the whole 5K-cycle burst, under various UR loads in the BE domain, is shown in Figure 5.17(b). Notice that the GB domain in PhaseNoC-OBS is completely immune to the load of the BE domain, and always manages to deliver the load in time, as needed for real-time traffic. In any case, in PhaseNoC-OBS the TDM pre-scheduled time slots at the VC level are gracefully combined with best-effort output port utilization to provide the best of both worlds.



(a) Real-time serving of GB traffic



(b) Sensitivity to BE load

Figure 5.17: Demonstration of the flow isolation properties of PhaseNoC-OBS. The Guaranteed-Bandwidth (GB) domain in PhaseNoC-OBS is completely immune to the load of the Best-Effort domain (BE), and always manages to deliver the GB traffic in time.

5.6 Related Work

At first glance, PhaseNoC may appear to be yet another facilitator of QoS guarantees. However, the proposed design is distinctly differentiated from existing techniques by its ability to *guarantee* full, secure-grade isolation between the supported domains. The service of communication flows *within each domain* could potentially be provided by any existing QoS technique. Hence, the PhaseNoC architecture is orthogonal to (and can benefit from) current mechanisms supporting differentiated services.

5.6.1 Priority-based Scheduling

Existing NoC techniques employing priority-based scheduling as a means to provide QoS guarantees were inspired by rate-based techniques in multi-computer networks, such as weighted fair queueing [35], virtual Clock [137], and rotating combined queueing [69]. Similarly, age-based arbitration prioritizes packets based on their age (oldest first) [3], while source throttling [124] can regulate flow rates to facilitate congestion control. Rate-based schemes employ fine-grained scheduling of packets, which incurs high implementation cost that is often prohibitive for on-chip implementations.

Hence, researchers have developed on-chip-amenable derivatives of rate-based approaches. Two examples are Globally Synchronized Frames (GSF) [79] and Preemptive Virtual Clock (PVC) [50], which utilize a coarse-grain rate-based philosophy based on the notion of time epochs, or “frames” [135]. By coarsening the granularity of bandwidth allocation, frame-based techniques reduce the hardware cost and scheduling complexity. To improve the efficiency of frame management, GSF [79] adopts an additional dedicated “barrier network,” which operates alongside the main NoC. Nevertheless, GSF still suffers from significant throughput degradation, as compared to a baseline router with no QoS provisioning. PhaseNoC could be augmented with GSF functionality (if desired), since each “frame” would correspond to a PhaseNoC domain. The GSF mechanism would control the traffic injection at the NIs, while GSF’s barrier network would monitor the global drainage of each domain. Similarly, non-linear weighted arbitration [52] can be incorporated *within* a PhaseNoC domain to shape the traffic within each domain and provide equality of service among the flows of a domain.

Unlike GSF’s reliance on multiple active frames, the PVC mechanism [50] employs only a single active frame at a time. However, PVC also relies on an additional network, which facilitates packet preemption (i.e., packet dropping and retransmission). In general, PVC suffers from higher implementation complexity than GSF, and it achieves markedly lower throughput than a baseline NoC. Furthermore, the PVC mechanism provides limited scalability with system size, as acknowledged in the topology-aware QoS schemes of [51].

Stall-time criticality enables application aware prioritization in the NoC [33] that improves system throughput without any QoS guarantees. Similarly, Aérgia [32] prioritizes in-flight packets based on how they affect the application’s execution time. LOFT [97] combines a frame-based approach and flit-reservation flow control [100] to improve network utilization over GSF. QNoC [22] provides four QoS service levels and uses preemptive scheduling to prioritize higher service levels. Per-priority-class buffering and preemptive scheduling at the flit-level are characteristics of any priority-based scheduling approach to avoid priority inversions.

5.6.2 TDM-based Scheduling

The vast majority of the TDM-based designs perform TDM scheduling at the time-slot level [46, 55, 66]. The scheduling is typically performed offline and then statically applied to the NoC. Argo [66] allows schedules to evolve at the granularity of a single cycle, even when the routers have more than one pipeline stage. The resulting hardware cost is quite low, but the latency overhead can be substantial.

Æthereal [46] employs pipelined TDM (at the time-slot level) and circuit-switching to guarantee performance services. Traffic is separated into two main classes: Guaranteed Service (GS) and Best Effort (BE). Excess bandwidth not used by GS flows is given to BE flows. Packets on a single connection are always ordered, but ordering cannot be enforced between connections. The utilization of excess bandwidth by other domains in PhaseNoC is facilitated by opportunistic bandwidth stealing. The SuperGT NoC [86] is an evolution of Æthereal providing three QoS classes. AElite simplifies the router architecture by providing only guaranteed service [55], and dAElite moves one step further by including multicast traffic and fast virtual-circuit setup [122]. TNoC relies on similar principles with aelite, providing only guaranteed service without the option of recycling unused schedule slots [114, 99]. The work of [129] provides run-time programmable bandwidth guarantees by regularly injecting epoch packets. Finally, Nostrum [91] and the work in [83] employ a form of TDM and virtual circuits to allocate bandwidth and satisfy QoS guarantees.

The overall simplicity of TDM-based NoCs, and their useful properties in providing QoS guarantees, are normally contrasted by the rather high buffering requirements. For example, TDM-based NoCs require virtual-circuit buffers (i.e., per connection) at least in the network interfaces, whereas PhaseNoC only requires buffers per application domain. In the same context, PhaseNoC-OBS mixes best-effort and guaranteed-service traffic, without resorting to additional virtual-circuit buffers for best effort connections [48]. PhaseNoC reuses VC buffers that normally refer to link-level flow control to statically define application domains as a generic mechanism for flow isolation. This reduces the need to dynamically set and tear down virtual circuits. The buffer savings expected from the use of VC buffers as isolated application domains – as opposed to using virtual circuits for the same purpose – is expected to be significant.

Additionally, PhaseNoC can be used in conjunction with TDM-based NoCs that apply TDM scheduling at the time-slot level. We believe that the TDM schedules derived for applying contention-free packet routing can be derived independently of their bandwidth guarantees, whereas the actual bandwidth allocation can be imposed at the VC-level by the PhaseNoC architecture. In this case, the algorithmic complexity of traditional TDM scheduling [57] is expected to be lower. This extension to PhaseNoC is also planned as future work.

5.6.3 Avoiding Flow/Traffic Interference in NoCs

One way to avoid interference among flows is to *physically separate* them in distinct networks. For example, the Tile64 iMesh separates user application traffic from OS and I/O traffic into separate physical networks [131]. Traffic isolation is also required in congestion management [39, 49]. ICARO [42] uses two virtual networks to isolate congested traffic from non-congested traffic, in order to avoid HoL blocking. PhaseNoC could provide similar functionality if combined with the congestion detection mechanism of [49] or [42].

Nevertheless, all approaches discussed so far may provide some QoS guarantees (e.g., minimum bandwidth and latency bounds) and/or differentiated service capabilitie, but none can *guarantee* complete flow isolation (non-interference). Many designs provide increased robustness to interference (e.g., PVC), but no absolute guarantees can be given, i.e., QoS properties are insufficient to guarantee non-interference within the network. [127].

Guaranteed non-interference has been investigated primarily within the realm of security [43]. In [127], static network partitioning in space and time is employed to provide multi-way isolation among the supported domains. This multi-way isolation property comes at a high performance cost, which is alleviated by the introduced RPSL mechanism, which uses priority-based arbitration and static limits to guarantee one-way isolation between high-security and low-security flows. The RPSL mechanism [127] facilitates bandwidth re-use among domains, just like PhaseNoC's OBS. However, its operation gives rise to fairness/starvation issues. As a result, RPSL requires an additional mechanism to monitor and statically limit the use of each input/output port by particular domains. On the contrary, PhaseNoC's OBS scheme enables bandwidth re-use without suffering from any fairness issues whatsoever.

Efficient multi-way non-interference among traffic flows is provided by SurfNoC [128] and TDM based architectures [46, 122]. Similar to PhaseNoC, SurfNoC also revolves around the concept of propagating waves across the on-chip network and TDM scheduling at the VC level. However, PhaseNoC is distinctly different and offers significant advantages over SurfNoC: (1) PhaseNoC's novel router micro-architecture allows each pipeline stage of the router to deal explicitly with the flits of only one domain. (2) Even though PhaseNoC achieves full isolation, it is less expensive (in terms of hardware) than even baseline designs, due to explicit pipelining, which allows for extensive sharing of the allocation logic. (3) Unlike SurfNoC, which always suffers a delay overhead on wave-front changes, PhaseNoC benefits from *zero-latency* propagation overhead, when satisfying the topology constraints derived in Section 5.3.1. The provided methodology generalizes to any network topology. (4) PhaseNoC jointly handles optimal scheduling and uneven allocation of bandwidth to the supported domains. (5) If secure-grade isolation is not required, PhaseNoC's OBS mechanism recycles any bandwidth left unused in each cycle to optimize performance.

5.7 Conclusions

In addition to providing high performance and scalability, NoCs may also be required to support additional functionality, such as flow isolation, and QoS provisioning. This paper introduces PhaseNoC, a cost-efficient VC-based NoC architecture supporting truly non-interfering multi-domain operation. The new design relies on TDM scheduling at the VC-level, which optimally coordinates the multi-phase propagation of domains across the NoC. PhaseNoC is a bandwidth-programmable traffic isolation mechanism, which is applicable to any topology upon following the proposed methodology for creating optimal phase schedules. When secure-grade domain isolation is not mandatory, PhaseNoC can also support a more "relaxed" isolation mode, which substantially improves the latency/throughput performance. By utilizing opportunistic bandwidth stealing, any unused bandwidth is recycled among the domains. Extensive evaluation using both synthetic traffic and real application workloads validates the efficiency of PhaseNoC, while detailed hardware analysis verifies the cost-effectiveness of the proposed new architecture.

Chapter 6

Conclusions

6.1 Summary

As the number of processing cores integrated on CMPs and SoCs continues to grow, addressing the on-chip communication demands with an efficient fabric becomes a major challenge. NoCs have been established as the dominant communication medium, due to their modular approach, their physical scalability, and ease of integration. To sustain future multi-core scaling without becoming the system's bottleneck, NoCs must provide high communication throughput and low-latency transfers, as well as specialized functionality, such as network virtualization, flow isolation, and QoS guarantees. These features must be provided in efficient hardware implementations that satisfy tight timing, area, and power constraints. This thesis addresses these fundamental challenges through four distinct, yet closely intertwined research directions, which collectively and synergistically weave a holistic approach to NoC design and implementation.

The first presented solution optimizes the micro-architecture of the NoC router. The aim is to achieve high-throughput and low-latency operation without exceeding the stringent area/energy constraints of modern SoCs, even when operating under a high clock frequency. Toward this end, we introduce ShortPath, a novel pipelined router architecture that can achieve high-speed implementations by parallelizing as much as possible – and without resorting to speculation – the allocation steps involved in the operation of a virtual-channel-based router. Additionally, ShortPath is augmented with a fine-grained pipeline bypassing mechanism, which skips all stages without contention and "fast-forwards" the flits to the first point of contention. Pipeline bypassing in ShortPath is always productive, and even if a flit loses in arbitration, it does not repeat any of the stages already bypassed. Consequently, ShortPath is the first (to the best of our knowledge) architecture to guarantee that each flit will spend the minimum possible number of cycles in each router, based on prevailing traffic conditions under both low and high network traffic.

In the second research strand comprising this thesis, we capitalize on the regularity in the physical layout of homogeneous tile-based CMPs to enable swift link traversal between neighboring tiles, after appropriate wire engineering. Specifically, we propose a technique to rapidly transfer flits between adjacent routers in half a clock cycle, by utilizing both edges of the clock during the sending and receiving operations. Half-cycle link traversal enables, for the first time, substantial reductions in (a) link power, irrespective of the data switching profile, and (b) buffer power (through buffer-size reduction), without incurring any latency/throughput loss. In fact, in addition to reaping substantial power savings, the presented architecture is also demonstrated to yield some latency improvements over a baseline NoC.

Building on the notion of half-cycle link traversal, the third solution presented in this thesis aims to improve network throughput without affecting the operating frequency of the NoC routers. The proposed RapidLink design harnesses the intrinsic asymmetry between the intra- and inter-router delays encountered in modern multicore systems, whereby the link traversal stage of a generic NoC router pipeline is substantially shorter than the delay encountered by flits within the router. This fast link traversal is utilized to enable DDR transfers between each upstream/downstream router pair, i.e., two flits can be sent/received per clock cycle. The only constraint imposed by RapidLink is that the link traversal delay cannot exceed one half of the delay of the router. However, RapidLink can also handle the cases where the link delay cannot fit within half of a clock cycle. In those cases, RapidLink "fragments" the link into multiple DDR half-cycle segments using newly introduced novel DDR dual-stream elastic buffers. In this way, the benefits of DDR link traversal can be reaped by any NoC design, even those with long wires and increased link delays. Most importantly, the adoption of RapidLink can markedly decrease latency and increase throughput, without incurring any additional hardware cost, as compared to traditional single-data-rate architectures. Viewed from a different perspective, DDR inter-router transfers can be used to extract increased throughput in environments where the technology and/or the power budget do not allow for higher operating frequencies.

After enabling high-throughput and low-latency communication, the thesis proceeds to augment the NoC infrastructure with additional functionality. Through a complete overhaul of the routers' micro-architecture and their internal pipeline operation, we enable secure-grade traffic isolation/non-interference, and flexible QoS provisioning with minimum-bandwidth and bounded-latency guarantees. The presented PhaseNoC paradigm is a truly non-interfering VC-based architecture that adopts time-division multiplexing at the Virtual Channel level. Distinct flows, or application domains, mapped to disjoint sets of VCs are isolated, both inside the router's pipeline and at the network level. Any latency overhead is minimized by appropriate scheduling of flows in separate phases of operation, irrespective of the chosen topology. When strict isolation is not required, the new architecture can employ opportunistic bandwidth stealing. This novel mechanism works in conjunction with the baseline PhaseNoC techniques to improve the overall latency/throughput characteristics of the NoC, while still preserving performance isolation.

The four architectures described in this thesis are conceptually orthogonal and they could potentially be combined to reap complementary benefits. For example, the PhaseNoC design is inherently amenable to "fuse" with RapidLink, because both mechanisms ensure flow isolation. Specifically, a PhaseNoC router can utilize and capitalize on the fact that RapidLink can interleave two independent streams on the link using the two opposite clock edges. Thus, two distinct PhaseNoC phases (traffic flows) can be interleaved on the link using a DDR RapidLink setup, without compromising PhaseNoC's isolation properties. Similarly, one can envision the augmentation of PhaseNoC functionality within the ShortPath router design. Incorporating – albeit in a modified manner – PhaseNoC's concept of explicit pipelining within ShortPath would allow for a highly-optimized pipelined router design that would also provide flow isolation and QoS guarantees.

6.2 Future Work

The architectures proposed in this thesis cover multiple aspects of NoC design, beginning from high-performance designs and energy-efficient approaches, and continuing to designs with increased functionality, that allow for network traffic isolation and secure switching. Although the presented architectures represent a mature set of optimized designs there are still many opportunities left for future improvements.

The first plan for extending the work presented in this thesis involves the incorporation in the Shortpath router architecture of the ability to schedule and forward multicast traffic. Supporting this feature is not straightforward, since Shortpath's pipelined and bypass functionality should be adapted accordingly, in order to prohibit the possible deadlock conditions that may arise during multicast scheduling. In any case, this effort would represent the first attempt ever in designing pipelined multicast-enabled routers that enable pipeline bypass in a non-speculative manner.

The second approach that seems as a natural evolution of the PhaseNoC architecture is the support for dynamically adjustable number of application domains. In the current version, the number of domains are statically decided at design-time on a per-application basis. In the extended version of PhaseNoC, we plan to enable the number of application domains to change dynamically, while still supporting network traffic isolation, both after the reconfiguration of the application domains, and during the transient phase of moving from one state to another.

The DDR NoCs presented in this thesis also open up many new possibilities for deriving even more efficient architectures. First of all, the baseline RapidLink NoC creates two independent streams that can be interleaved on the link using opposite edges of the clock. This allows for DDR operation on the links without increasing the clock frequency of the partitioned NoC routers. Even if we showed how this interleaving may work for single-stream applications, a more generic approach should be derived. Our goal is to generalize the latency-insensitive protocol (ready/valid) and the elastic buffers that implement it to the case of dual-edge clocking. By sampling data on both the rising and falling edges of the clock, the clock frequency can be reduced by 50%, without changing the system throughput. This directly cuts the power dissipation of the clock network in half, leading to significant overall system power savings.

In this way, we increase the applicability of the dual-edge-triggered clocking and effectively reduce the clock-tree power of NoCs. Both alternatives – loosely connected up to now – have been proposed independently as a means to tackle the scaling of the clock frequencies and interconnect latencies of modern SoCs. With multi-GHz clock periods and significant on-chip variability, the importance of clock uncertainties will increase. Thus, the dual-edge-triggered clocking strategy alleviates frequency scaling problems by retaining the data throughput of single-edge-triggered clocking at half the clock frequency.

Finally, technology scaling, process variations, and/or 3D integration make the design of fully synchronous SoCs a challenging task. Partitioning the SoC into Globally Asynchronous, Locally Synchronous (GALS) islands partially alleviates the difficulties in clock distribution. Such partitioning of the SoC is also necessary when supporting Dynamic Voltage and Frequency Scaling (DVFS) across parts of the system to minimize power consumption. The NoC is an inherently distributed architecture that is physically spread over the entire chip; thus, it should readily support communication across multiple clock domains. Any clock-domain crossing interface should also seamlessly support the flow control rules that govern data transfer on every NoC link.

In the future, we plan to generalize the fundamental properties of VC flow control across asynchronous and mesochronous clock domains. In parallel, our goal is to also generalize the design of dynamically-allocated multiple-VC shared buffers, and propose dual-clock alternatives, whereby the read and write ports can operate in different clock domains. Our intent is for the derived dual-clock shared buffer to merge the tasks of signal synchronization across asynchronous clock domains, and queueing (buffering), in a common hardware module. In this way, the envisioned tightly-coupled approach would offer maximum scalability in terms of latency, throughput, and buffering requirements.

Bibliography

[1] N. Abeyratne, R. Das, Q. Li, K. Sewell, B. Giridhar, R. Dreslinski, D. Blaauw, and T. Mudge. Scaling toward kilo-core processors with asymmetric high-radix topologies. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[2] Dennis Abts and Bob Felderman. A guided tour of datacenter networking. *Communications of the ACM - ACM Queue*, 55(6):44–51, 2012.

[3] Dennis Abts and Deborah Weisser. Age-based packet arbitration in large k-ary n-cubes. In *ACM/IEEE Proc. of the Supercomputing Conference (SC)*, pages 1–11, 2007.

[4] Accelera. OCP-IP protocol specification, v3.0. 2013.

[5] N. Agarwal, T. Krishna, L.S. Peh, and N.K. Jha. GARNET: A detailed on-chip network model inside a full-system simulator. In *Proc. of the IEEE Intern. Symp. on Performance Analysis of Systems and Software*, pages 33–42, April 2009.

[6] Ittai Anati, David Blythe, Jack Doweck, Hong Jiang, Wen-fu Kao, Julius Mandelblat, Lihu Rappoport, Efraim Rotem, and Ahmad Yasin. Inside 6th gen intel® core: New microarchitecture code named skylake. In *Hot Chips 28 Symposium (HCS), 2016 IEEE*, pages 1–39. IEEE, 2016.

[7] ARM. AMBA AXI and ACE Protocol Specification, 2013.

[8] Arteris. A comparison of network-on-chip and buses. Technical report, 2005.

[9] Krste Asanovic and et al. The landscape of parallel computing: The view from berkeley, Dec. 2006. Technical Report No. UCB/EECS-2006-183.

[10] IEEE Standards Association. IEEE standard for systemc language reference manual. *IEEE Std. 1666-2011*, 2011.

[11] IEEE Standards Association. IEEE standard for universal verification methodology language reference manual. *IEEE Std. 1800.2-2017*, 2017.

[12] M. Azimi et al. Flexible and adaptive on-chip interconnect for tera-scale architectures. *Intel Technology Journal*, (4):62–77, 2009.

[13] M. Azimi et al. *On-chip Interconnect Trade-offs for Tera-scale Many-core Processors*. Designing Network On-Chip Architectures in the Nanoscale Era, Jose Flich and Davide Bertozzi, Eds., CRC Press, 2011.

[14] Ali Bakhoda, John Kim, and Tor M. Aamodt. Throughput-effective on-chip networks for manycore accelerators. In *Proc. of Intern. Symposium on Microarchitecture (MICRO)*, pages 421–432, 2010.

[15] James Balfour and William J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Proceedings of the 20th ACM International Conference on Supercomputing (ICS)*, pages 187–198. ACM, June 2006.

[16] Daniel Becker. *Efficient microarchitecture for network-on-chip routers*. PhD thesis, Stanford University, 2012.

[17] Daniel U Becker, Nan Jiang, George Michelogiannakis, and William J Dally. Adaptive backpressure: Efficient buffer management for on-chip networks. In *Intern. Conf. on Computer Design*, pages 419–426, 2012.

[18] Luca Benini and Giovanni De Micheli. Networks on chips: A new soc paradigm. *Computer*, 35(1):70–78, January 2002.

[19] J. Bhasker and C. Rakesh. *Static timing analysis for nanometer designs: A practical approach*. Springer Science & Business Media, 2009.

[20] C. Bienia et al. The PARSEC benchmark suite: Characterization and architectural implications. In *Proc. of Int. Conf. on Parallel Archs. and Compilation Techniques*, pages 72–81, October 2008.

[21] T. Bjerregaard, M. Stensgaard, and J. Sparso. A scalable, timing-safe, network-on-chip architecture with an integrated clock distribution method. In *Design Automation and Test in Europe (DATE)*, 2007.

[22] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. QNoC: QoS architecture and design process for network on chip. *Journal of systems architecture*, 50(2):105–128, 2004.

[23] S. Borkar and A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, May 2011.

[24] Yuan-Ying Chang et al. Ts-router: On maximizing the quality-of-allocation in the on-chip network. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 390–399, 2013.

[25] Jordi Cortadella, Mike Kishinevsky, and Bill Grundmann. Synthesis of Synchronous Elastic Architectures. In *ACM Design Automation Conference (DAC)*, pages 657–662, July 2006.

[26] William J. Dally. Virtual-Channel Flow Control. In *ISCA*, pages 60–68, 1990.

[27] William J. Dally. Virtual-channel flow control. *IEEE Trans. on Parallel and Distributed Systems*, 3(3):194–205, March 1992.

[28] William J. Dally, Chris Malachowsky, and Stephen W. Keckler. 21st century digital design tools. In *ACM Design Automation Conference (DAC)*, DAC '13, pages 94:1–94:6, 2013.

[29] William J. Dally and Brian Towles. Route packets, not wires: On-chip inteconnection networks. In *Proc. of the 38th Annual Design Automation Conference*, pages 684–689, 2001.

[30] William J. Dally and Brian Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proc. of the 38th Design Automation Conference (DAC)*, June 2001.

[31] William J. Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[32] Reetuparna Das et al. Aérgia: exploiting packet latency slack in on-chip networks. In *ACM SIGARCH Comp. Arch. News*, volume 38, pages 106–116, 2010.

[33] Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R Das. Application-aware prioritization mechanisms for on-chip networks. In *Proc. 42nd IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, pages 280–291, 2009.

[34] B.K. Daya, C.-H.O. Chen, S. Subramanian, K. Woo-Cheol, P. Sunghyun, T. Krishna, J. Holt, A. P. Chandrakasan, and L.S. Peh. Scorpio: A 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering. In *International Symposium on Computer Architecture*, pages 25–36, June 2014.

[35] Alan Demers et al. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM*, volume 19, pages 1–12, 1989.

[36] G. Dimitrakopoulos, A. Psarras, and I. Seitanidis. *Microarchitecture of Network-on-Chip Routers: A designer's perspective*. Springer, 2014.

[37] Giorgos Dimitrakopoulos, Emmanouil Kalligeros, and Kostas Galanopoulos. Merged switch allocation and traversal in network-on-chip switches. *IEEE Transactions on Computers*, 62(10):2001–2012, 2013.

[38] José Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1320–1331, 1993.

[39] Jose Duato et al. A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks. In *Proc. of High-Performance Computer Architecture*, pages 108–119, 2005.

[40] Yves Durand, Christian Bernard, and Didier Lattard. Faust: On-chip distributed architecture for a 4g baseband modem soc. *Proceedings of Design and Reuse IP-SOC*, 5:51–55, 2005.

[41] M. Ebrahimi et al. A high-performance network interface architecture for nocs using reorder buffer sharing. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 546–550. IEEE, 2010.

[42] Jose V Escamilla et al. ICARO: Congestion isolation in networks-on-chip. In *IEEE Int. Symp. on Networks-on-Chip*, pages 159–166, 2014.

[43] Leandro Fiorin et al. A security monitoring service for NoCs. In *Hardware/Software codesign and system synthesis*, pages 197–202, 2008.

[44] M. Galles. Spider: A high-speed network interconnect. *IEEE Micro*, 17(1), 1997.

[45] A. Golander, N. Levison, O. Heymann, A. Briskman, M. J. Wolski, and E. F. Robinson. A cost-efficient L1–L2 multicore interconnect: Performance, power, and area considerations. *IEEE Transactions on Circuits and Systems-I: Regural Papers*, 58(3):529–538, March 2011.

[46] K. Goossens et al. Æthereal network on chip: concepts, architectures, and implementations. *IEEE Design & Test*, 22:414–421, 2005.

[47] Kees Goossens et al. Virtual execution platforms for mixed-time-criticality systems: The CompSOC architecture and design flow. *ACM SIGBED Rev.*, 10(3):23–34, October 2013.

[48] Kees Goossens and Andreas Hansson. The aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *Design Automation Conference (DAC)*, pages 306–311, 2010.

[49] Paul Gratz, Boris Grot, and Stephen W Keckler. Regional congestion awareness for load balance in networks-on-chip. In *Proc. 14th IEEE Int. Symp. on High Performance Computer Architecture (HPCA)*, pages 203–214, 2008.

[50] B. Grot et al. Preemptive virtual clock: a flexible, efficient, and cost-effective QoS scheme for networks-on-chip. In *Proc of Intern. Symposium on Microarchitecture (MICRO)*, pages 268–279, 2009.

[51] B. Grot et al. Kilo-NOC: a heterogeneous network-on-chip architecture for scalability and service guarantees. In *ISCA*, pages 401–412, 2011.

[52] K. Gwangsun, M. M. J. Lee, J. Kim, D. Abts, and M. Marty. Low-overhead network-on-chip support for location-oblivious task placement. *IEEE Trans. on Computers*, 63(6):1487–1500, 2014.

[53] By Linley Gwennap. Adapteva: More flops, less watts epiphany offers floating-point accelerator for mobile processors. *Microprocessor Report*, 2011.

[54] Jim Handy. NoC interconnect improves SoC economics. *Objective analysis - Semiconductor market research*, 2011.

[55] A. Hansson et al. aelite: A flit-synchronous network on chip with composable and predictable services. In *DATE*, pages 250–255, 2009.

[56] A. Hansson, K. Goossens, and A. Rădulescu. Avoiding message-dependent deadlock in network-based systems on chip. *VLSI design*, 2007, 2007.

[57] Andreas Hansson, Kees Goossens, and Andrei Radulescu. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Proc. of Intern. Conf. on Hardware/Software Codesign and System Synthesis*, pages 75–80, 2005.

[58] Syed Minhaj Hassan and Sudhakar Yalamanchili. Centralized buffer router: A low latency, low power router for high radix nocs. In *Networks on Chip (NoCS), 2013 Seventh IEEE/ACM International Symposium on*, pages 1–8. IEEE, 2013.

[59] Byungchul Hong, Brian Huang, and Jonah Probell. The rubber jigsaw puzzle floorplanning for network-on-chip. In *Synopsys Users Group Conference (SNUG)*, 2015.

[60] Yatin Hoskote, Sriram Vangal, Arvind Singh, Nitin Borkar, and Shekhar Borkar. A 5-GHz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5):51–61, 2007.

[61] Yuanjie Huang, Paolo Ienne, Olivier Temam, Yunji Chen, and Chengyong Wu. Elastic CGRAs. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA 2013*, 2013.

[62] H. M. Jacobson et al. Synchronous interlocked pipelines. In *Proc. International Symposium on Asynchronous Circuits and Systems*, pages 3–12, April 2002.

[63] Doddaballapur Jayasimha, Jeremy Chan, and Jay Tomlison. Use of common data format to facilitate link width conversion in a router with flexible link widths, 2011.

[64] Doddaballapur N Jayasimha, Jeremy Chan, and Jay S Tomlinson. Use of common data format to facilitate link width conversion in a router with flexible link widths, August 20 2013. US Patent 8,514,889.

[65] T. Karnik and et al. Total power optimization by simultaneous dual-vt allocation and device sizing in high performance microprocessors. In *Proceedings of the 39th Annual Design Automation Conference*, DAC '02, pages 486–491, 2002.

[66] Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Srensen, Christian T. Mller, Kees Goossens, and Jens Sparso. Argo: A real-time network-on-chip architecture with an efficient gals implementation. *in IEEE Trans. on VLSI Systems,*, 2015.

[67] Manolis Katevenis, Stefanos Sidiropoulos, and Costas Courcoubetis. Weighted round-robin cell multiplexing in a general-purpose atm switch chip. *IEEE Journal on Selected Areas in Communications*, pages 1265–1279, 1991.

[68] J. Kim, J. Balfour, and W. J. Dally. Flattened butterfly topology for on-chip networks. In *MICRO 40: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 172–182, Washington, DC, USA, 2007. IEEE Computer Society.

[69] Jae H Kim and Andrew A Chien. Rotating combined queueing (RCQ): bandwidth and latency guarantees in low-cost, high-performance networks. In *ACM SIGARCH Comp. Arch. News*, volume 24, pages 226–236, 1996.

[70] Avinash Karanth Kodi, Ashwini Sarathy, and Ahmed Louri. ideal: Inter-router dual-function energy and area-efficient links for network-on-chip (noc) architectures. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 241–250. IEEE Computer Society, 2008.

[71] Tushar Krishna, Chia-Hsin Owen Chen, Woo Cheol Kwon, and Li-Shiuan Peh. Smart: Single-cycle multi-hop traversals over a shared network-on-chip. *IEEE Micro*, May/June 2014.

[72] Tushar Krishna, Chia-Hsin Owen Chen, Sunghyun Park, Woo-Cheol Kwon, Suvinay Subramanian, Anantha P. Chandrakasan, and Li-Shiuan Peh. Single-cycle multihop asynchronous repeated traversal: A smart future for reconfigurable on-chip networks. *IEEE Computer*, 2013.

[73] Tushar Krishna and Li-Shiuan Peh. Single-cycle collective communication over a shared network fabric. In *International Symposium on Networks-on-Chip (NOCS)*, 2014.

[74] Tushar Krishna, Jacob Postman, Christopher Edmonds, Li-Shiuan Peh, and Patrick Chiang. Swift: A swing-reduced interconnect for a token-based network-on-chip in 90nm cmos. In *Computer Design (ICCD), 2010 IEEE International Conference on*, pages 439–446. IEEE, 2010.

[75] Amit Kumar, Partha Kundu, Arvind Singh, Li-Shiuan Peh, and Niraj K. Jha. A 4.6tbits/s 3.6ghz single-cycle noc router with a novel switch allocator in 65nm cmos. In *Proc. of the Intern. Conf. on Computer Design*, pages 63–70, Oct. 2007.

[76] Amit Kumar, Li-Shiuan Peh, and Niraj K Jha. Token flow control. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, pages 342–353. IEEE Computer Society, 2008.

[77] Prabhat Kumar, Yan Pan, John Kim, Gokhan Memik, and Alok Choudhary. Exploring concentration and channel slicing in on-chip network router. In *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '09, pages 276–285, 2009.

[78] Jean-Jacques Lecler and Philippe Boucard. Zero-latency network on chip (NoC), 2011.

[79] J. W. Lee et al. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. In *ISCA*, pages 89–100, 2008.

[80] Junghee Lee, Chrysostomos Nicopoulos, Hyung Gyu Lee, and Jongman Kim. TornadoNoC: A lightweight and scalable on-chip network architecture for the many-core era. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4), Dec. 2013.

[81] Junghee Lee, Chrysostomos Nicopoulos, Sung Joo Park, Madhavan Swaminathan, and Jongman Kim. Do we need wide flits in networks-on-chip? In *VLSI (ISVLSI), 2013 IEEE Computer Society Annual Symposium on*, pages 2–7. IEEE, 2013.

[82] Ye Lu, Changlin Chen, John V. McCanny, and Sakir Sezer. Design of interlock-free combined allocators for networks-on-chip. In *EEE 25th International SOC Conference (SoCC)*, pages 358–363, 2012.

[83] Zhonghai Lu and Axel Jantsch. TDM virtual-circuit configuration for network-on-chip. *IEEE Trans. on VLSI*, 16(8):1021–1034, 2008.

[84] Nancy D MacDonald. Timing closure in deep submicron designs. *DAC. com Knowledge Center Article*, 2010.

[85] Ran Manevich, Leon Polishuk, Israel Cidon, and Avinoam Kolodny. Designing single-cycle long links in hierarchical nocs. *Microprocessors and Microsystems*, 38(8):814 – 825, 2014.

[86] Théodore Marescaux and Henk Corporaal. Introducing the SuperGT network-on-chip; SuperGT QoS: More than just GT. In *Proc. 44th ACM/IEEE Design Automation Conference (DAC)*, pages 116–121, 2007.

[87] Milo M. K. Martin and et al. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, November 2005.

[88] Hiroki Matsutani, Michihiro Koibuchi, Hideharu Amano, and Tsutomu Yoshinaga. Prediction router: A low-latency on-chip router architecture with multiple predictors. *IEEE Transactions on Computers*, 60(6):783–799, 2011.

[89] G. Michelogiannakis, N.Jiang, D.Becker, and W.J.Dally. Packet chaining: Efficient single-cycle allocation for on-chip networks. In *Proc. Intern. Symp. on Microarchitecture*, pages 83–94, 2011.

[90] George Michelogiannakis and William J Dally. Elastic buffer flow control for on-chip networks. *IEEE Transactions on Computers*, 62(2):295–309, 2013.

[91] Mikael Millberg and et al. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum

network on chip. In *Design Automation and Test in Europe Conference*, pages 890–895, 2004.

[92] Ivan Miro-Panades, Fabien Clermidy, Pascal Vivet, and Alain Greiner. Physical implementation of the dspin network-on-chip in the faust architecture. In *International Symposium on Networks-on-Chip*, pages 139–148, 2008.

[93] Robert Mullins, Andrew West, and Simon Moore. Low-latency virtual-channel routers for on-chip networks. In *Proceedings of the International Symposium on Computer Architecture*, pages 188–197. IEE, June 2004.

[94] S. Murali et al. Designing message-dependent deadlock free networks on chips for application-specific systems on chips. In *Very Large Scale Integration, 2006 IFIP International Conference on*, pages 158–163. IEEE, 2006.

[95] C. Nicopoulos, S. Srinivasan, A. Yanamandra, D. Park, V. Narayanan, C. Das, and M. J. Irwin. On the effects of process variation in network-on-chip architectures. *IEEE Trans. Dependable Sec. Comput.*, 7(3):240–254, 2010.

[96] Andreas Olofsson. Epiphany-v: A 1024 processor 64-bit risc system-on-chip. *arXiv preprint arXiv:1610.01832*, 2016.

[97] Jin Ouyang and Yuan Xie. LOFT: A high performance network-on-chip providing quality-of-service support. In *Proc. 43rd IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, pages 409–420, 2010.

[98] P. Salihundam et al. A 2Tb/s 6x4 Mesh Network with DVFS and 2.3Tb/s/W router in 45nm CMOS. In *VLSI Circuits*, 2010.

[99] C. Paukovits and Hermann Kopetz. Concepts of switching in the time-triggered network-on-chip. In *14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 120–129. IEEE, 2008.

[100] Li-Shiuan Peh and William J Dally. Flit-reservation flow control. In *Proc. of High-Performance Computer Architecture (HPCA)*, pages 73–84, 2000.

[101] Li-Shiuan Peh and William J. Dally. A delay model and speculative architecture for pipelined routers. In *Proc. Intern. Symp. on High-Performance Computer Architecture*, pages 255–266, January 2001.

[102] J. Philip, J. Rowlands, and S. Kumar. Multiple clock domains in noc, December 25 2014. US Patent App. 13/922,053.

[103] Jacob Postman, Tushar Krishna, Christopher Edmonds, Li-Shiuan Peh, and Patrick Chiang. Swift: A low-power network-on-chip implementing the token flow control router architecture with swing-reduced interconnects. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(8):1432–1446, Aug. 2012.

[104] A. Psarras, J. Lee, P. Mattheakis, C. Nicopoulos, and G. Dimitrakopoulos. A low-power network-on-chip architecture for tile-based chip multi-processors. In *ACM Great Lakes Symp-VLSI*, pages 335–340, 2016.

[105] A. Psarras, J. Lee, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos. Phasenoc: Versatile network traffic isolation through tdm-scheduled virtual channels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(5):844–857, 2016.

[106] A. Psarras, S. Moisidis, C. Nicopoulos, and G. Dimitrakopoulos. Rapidlink: A network-on-chip architecture with double-data-rate links. In *Electronics, Circuits and Systems (ICECS), 2016 IEEE International Conference on*, pages 93–96. IEEE, 2016.

[107] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos. PhaseNoC: TDM scheduling at the virtual-channel level for efficient network traffic isolation. In *Proceedings of the 2015 Design, Automation & Test in Europe*, pages 1090–1095, 2015.

[108] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos. ShortPath: A network-on-chip router with fine-grained pipeline bypassing. *IEEE Transactions on Computers*, 65(10):3136–3147, 2016.

[109] A. Pullini, F. Angiolini, S. Murali, D. Atienza, G. De Micheli, and L. Benini. Bringing nocs to 65 nm. *IEEE Micro*, 27(5):75–85, Sept. 2007.

[110] Andrei Radulescu, John Dielissen, Santiago González Pestana, Om Prakash Gangwal, Edwin Rijpkema, Paul Wielage, and Kees

Goossens. An efficient on-chip ni offering guaranteed services, shared-memory abstraction, and flexible network configuration. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 24(1):4–17, 2005.

[111] Carl Ramey. Tile-gx100 manycore processor: Acceleration interfaces and architecture. In *Hot Chips 23 Symposium (HCS), 2011 IEEE*, pages 1–21. IEEE, 2011.

[112] Xavier Van Ruymbeke. Benefits of network on chip fabrics for late stage design changes, adaptive qos and floorplan selection. In *ChipEX*, April 2014.

[113] Sergio Saponara, Luca Fanucci, and Marcello Coppola. Design and coverage-driven verification of a novel network-interface ip macrocell for network-on-chip interconnects. *Microprocessors and Microsystems*, 35(6):579–592, 2011.

[114] Martin Schoeberl. A time-triggered network-on-chip. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 377–382. IEEE, 2007.

[115] I. Seitanidis, A. Psarras, K. Chrysanthou, C. Nicopoulos, and G. Dimitrakopoulos. Elastistore: Flexible elastic buffering for virtual-channel-based networks on chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(12):3015–3028, 2015.

[116] I. Seitanidis, A. Psarras, G. Dimitrakopoulos, and C. Nicopoulos. ElastiStore: An elastic buffer architecture for network-on-chip routers. In *IEEE Design, Automation and Test in Europe (DATE)*, Mar. 2014.

[117] I. Seitanidis, A. Psarras, E. Kalligeros, C. Nicopoulos, and G. Dimitrakopoulos. ElastiNoC: A self-testable distributed vc-based network-on-chip architecture. In *International Symposium on Networks-on-Chip (NOCS)*, pages 135–142, 2014.

[118] Erik Seligman, Tom Schubert, and MV Achutha Kiran Kumar. *Formal verification: an essential toolkit for modern VLSI design*. Morgan Kaufmann, 2015.

[119] Jae Sun Seo, Dennis Sylvester, David Blaauw, Himanshu Kaul, and Ram Krishnamurthy. A robust edge encoding technique for

energy-efficient multi-cycle interconnect. In *ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, August 2007.

[120] K. Sewell et al. Swizzle-switch networks for many-core systems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(2):278–294, 2012 2012.

[121] Avinash Sodani. Knights landing (knl): 2nd generation intel® xeon phi processor. In *Hot Chips 27 Symposium (HCS), 2015 IEEE*, pages 1–24. IEEE, 2015.

[122] Radu Stefan, Anca Molnos, and Kees Goossens. daelite: A tdm noc supporting qos, multicast, and fast connection set-up. *IEEE Transactions on Computers*, 63(3):583–594, March 2014.

[123] Vladimir Stojanovic. Design of energy-efficient on-chip networks. In *ISSCC Tutorial*, 2010.

[124] Mithuna Thottethodi et al. Self-tuned congestion control for multiprocessor networks. In *High-Performance Comp. Architecture*, pages 107–118, 2001.

[125] Mohit Tiwari et al. Complete information flow tracking from the gates up. In *Proc. of Intern. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 109–120, 2009.

[126] Sriram R Vangal et al. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *IEEE Journal of solid-state circuits*, 43(1):29–41, 2008.

[127] Yao Wang and G Edward Suh. Efficient timing channel protection for on-chip networks. In *Proc. 6th IEEE/ACM Int. Symp. on Networks on Chip (NoCS)*, pages 142–151, 2012.

[128] H. Wassel et al. SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip. In *ISCA*, pages 583–594, 2013.

[129] W-D Weber, Joe Chou, Ian Swarbrick, and Drew Wingard. A quality-of-service mechanism for interconnection networks in system-on-chips. In *Design, Automation and Test in Europe (DATE)*, pages 1232–1237, 2005.

[130] Wolf-Dietrich Weber, Joseph Harwood, Michael Meyer, and Drew Wingard. Various methods and apparatuses for width and burst conversion, June 2 2009. US Patent 7,543,088.

[131] David Wentzlaff et al. On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro*, pages 15–31, Sep./Oct. 2007.

[132] Q. Wu, M. Pedram, and X. Wu. A new design of double edge triggered flip-flops. In *Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pages 417–421, 1998.

[133] Xu Yang, Zhang Qing-li, Fu Fang-fa, Yu Ming-yan, and Liu Cheng. NISAR: An AXI compliant on-chip NI architecture offering transaction reordering processing. In *ASIC, 2007. ASICON'07. 7th International Conference on*, pages 890–893. IEEE, 2007.

[134] Young Jin Yoon, Nicola Concer, Michele Petracca, and Luca P Carloni. Virtual channels and multiple physical networks: Two alternatives to improve noc performance. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 32(12):1906–1919, 2013.

[135] Hui Zhang and Srinivasan Keshav. Comparison of rate-based service disciplines. In *ACM SIGCOMM Computer Communication Review*, volume 21, pages 113–121, Sept. 1991.

[136] Jiayi Zhang. Design and implementation of axi-based network-on-chip systems for flow regulation, 2009.

[137] Lixia Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. In *ACM SIGCOMM*, pages 19–29, 1990.