Virtualizing Virtual Channels for Increased Network-on-Chip Robustness and Reliability

Marios Evripidou
Department of Electrical and Computer Engineering
University of Cyprus

A thesis submitted for the degree of
Master of Science (MSc) in Computer Engineering
2011 December

Committee Members:

Chrysostomos Nicopoulos
Lecturer, Department of ECE, Advisor

Maria K. Michael
Assistant Professor, Department of ECE

Theocharis Theocharides
Assistant Professor, Department of ECE

Date of the defence: 8th of December 2011

Abstract

The Network-on-Chip (NoC) router buffers are instrumental in the overall operation of Chip Multi-Processors (CMP), because they facilitate the creation of Virtual Channels (VC). Both the NoC routing algorithm and the CMPs cache coherence protocol rely on the presence of VCs within the NoC for correct functionality. The router buffer space is partitioned a priori at design-time and the resulting NoC has a fixed number of VCs in each router port. This characteristic raises two critical concerns stemming from its innate inflexibility: (1) System functionality in the event of a VC buffer-/channel malfunction: a malfunction in any of the VC components may lead to network and/or cache coherence protocol deadlocks. (2) System upgradeability with new routing algorithms, and/or new cache coherence protocols, which require different numbers of VCs: Statically partitioned VC implementations cannot accommodate routing algorithms and cache coherence protocols that require different number of VCs.

This pair of problematic facets in conventional VC implementations serves as the primary driver of the work presented in this thesis: Re-engineering the VC operation so as to achieve both robustness and the ability to support multiple routing algorithms and cache coherence protocols with no restrictions on the number of supported VCs. The key idea is to enable the further virtualization of virtual channels. Through this process, the system would allow the mapping of any number of Virtual Virtual Channels (VVC) on top of the existing virtual channel buffers (Physical Virtual Channels, or PVC). The proposed technique (Virtual Channel Renaming - VC Renaming) decouples the number of VCs (now known as VVCs) required by the routing algorithm and/or the cache coherence protocol from the actual number of physical VCs (PVCs) originally built into the system at design-time. The notion of virtualizing virtual channels is an innovative idea, which, to the best of our knowledge, has not been suggested in the literature before. Our simulation results are very promising, and hardware synthesis using commercial 65 nm TSMC libraries demonstrates modest area/power overhead and no impact on the routers critical path.

To my family and friends

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Prof. Chrysostomos Nicopoulos for the continuous support during my MSc study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my MSc study. I would also like to thank Prof. Vassos Soteriou for his valuable contribution in the early stages of this project and his continuous assistance during the development of this work.

I would also like to thank my friends for sticking by me all these years and always reminding me that some things are much more precious than others. Without them I wouldn't be who I am today.

Last but not the least, I would like to thank my family for raising me and giving me the values I uphold through life, for always believing in me and supporting me spiritually throughout my life.

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

1

Introduction

The advent of Chip Multi-Processors (CMPs) has accentuated the criticality of the on-chip communication infrastructure, which is now tasked with the mission-critical objective of maintaining swift and reliable inter-core communication. Escalating numbers of on-chip processing cores necessitate the introduction of an efficient and scalable communication backbone. Packet-based Networks-on-Chip (NoC)(2, 3, 4, 5) are envisioned as the most viable solution for the many-core chips of the near future.

Among the various components comprising the on-chip network backbone, the router buffers constitute one of the fundamental cogs in the operation of the Networks-on-Chip. The buffering resources orchestrate the flow-control mechanism of the network and facilitate the so called Virtual Channels (VCs), which enable the multiplexing of several packets onto a single physical channel. More importantly, however, virtual channels are obligatory constructs for the correct functionality of two elemental operations within the Chip Multi-Processors:

(a) Network Routing Computation: The vast majority of *adaptive* routing algorithms developed for interconnection networks rely on the extensive use of Virtual Channels for deadlock avoidance and/or deadlock recovery. Existing *adaptive* routing algorithms require anywhere from 2 to 16 virtual channels per router input port for functional correctness (6). While *deterministic* routing algorithms (e.g., XY routing) may require only *one* virtual channel, adaptive algorithms offer much more flexibility, adaptability to prevailing traffic conditions, and resilience to faults.

(b) Cache Coherence: The plethora of cache coherence protocols designed for modern Chip Multi-Processors necessitate the use of multiple virtual networks (realized through virtual channels), in order to avoid protocol deadlocks. For instance, the well-known MOESI protocol requires 3 virtual networks (i.e., at least 3 Virtual Channels), while other protocols require anywhere from 2 to 8 virtual channels per router input port (7).

Hence, virtual channels are key enablers for the seamless functionality of both the network fabric and the cache/memory sub-system of multi-core microprocessors.

Most existing Networks-on-Chip designs employ statically partitioned virtual channel resources. In other words, the router buffer space is partitioned a priori at design-

time (i.e., pre-manufacturing) and the resulting Networks-on-Chip has a fixed number of virtual channels in each router port. This characteristic raises two critical concerns stemming from its innate inflexibility:

(1) System functionality in the event of a virtual channel buffer/channel malfunction – The issue of hardware reliability is becoming increasingly relevant as technology scales deep into the nano-scale regime (8, 9). In general, faults (permanent, intermittent, and transient) may occur anywhere on the chip and may affect any system component. Defects in various locations within the Networks-on-Chip may affect the functionality of virtual channels directly (e.g., buffer faults), or indirectly (e.g., arbiter faults). The Network-on-Chip virtual channels are so deeply intertwined with the operation of the Chip Multi-Processors, that a malfunction in any of the virtual channel components may lead to network and/or cache coherence protocol deadlocks (i.e., whole-system inoperability).

(2) System upgradeability with new routing algorithms, and/or new cache coherence protocols, which require different numbers of virtual channels – A major limitation with a statically partitioned virtual channel implementation is the inability to accommodate routing algorithms and cache coherence protocols that require a different number of virtual channels than the specific (and fixed) number of virtual channels ingrained into the Chip Multi-Processor at design-time. The concept of *upgradeability* is not merely academic. The incessant emergence of newer and more demanding *applications* may require the introduction of newer (and/or *customized*) routing algorithms, in order to optimize system performance. Such capability is currently limited by the rigid requirement of strict compliance with the existing number of virtual channels in the Networks-on-Chip.

This pair of problematic facets in conventional virtual channel implementations serves as the primary driver of the work presented in this thesis: **Re-engineering the virtual channel operation** so as to achieve both **robustness** and the ability to **support multiple routing algorithms and cache coherence protocols** with *no restrictions on the number of supported virtual channels*. Toward this end, we hereby introduce the concept of **Virtual Channel Renaming (VC Renaming)**. The key idea is to enable the further ***virtualization of virtual channels***. Through this process, the system would allow the mapping of any number of *Virtual Virtual Channels (VVCs)* on top of the existing (i.e., statically partitioned at design-time) virtual channel buffers (henceforth called *Physical Virtual Channels*, or *PVCs*). In essence, the technique of VC Renaming facilitates the creation of an arbitrary number of Virtual Virtual Channels (where the *Number of Virtual Virtual Channels* $\geq$ *Number of Physical Virtual Channels*), irrespective of the original fixed number of physical virtual channels built into the Networks-on-Chip at manufacturing.

For example, suppose the Networks-on-Chip of a multi-core microprocessor is implemented with 3 virtual channels (i.e., Number of physical virtual channels = 3) per router input port. If one of the virtual channel buffers in any router's input port malfunctions, the VC Renaming engine will map the affected virtual channel onto one of the remaining 2 (fully-functioning) physical virtual channels. Similarly, if a new routing

algorithm is to be used, which requires 4 virtual channels per input port, an additional virtual virtual channel will be mapped onto one of the 3 physical virtual channels, so that a total of 4 virtual virtual channels will be served by the existing 3 physical virtual channels.

The proposed VC Renaming technique decouples the number of virtual channels (now known as virtual virtual channels) required by the routing algorithm and/or the cache coherence protocol from the actual number of physical virtual channels (PVCs) originally built into the system at design-time. The mechanism is completely transparent to the neighboring Networks-on-Chip routers: the routers are only aware of the presence of a specific number of virtual virtual channels (those required by the routing algorithm and/or the coherence protocol). A disabled phyrical virtual channel in one router will not become visible to the neighbors; VC Renaming will hide the anomaly by remapping the affected virtual virtual channel onto a different physical virtual channel. System functionality will be fully maintained, albeit at a degraded performance. The principle of VC Renaming is inspired by the well-known architectural technique of Register Renaming, which overcomes the restriction of a fixed number of architectural registers within the CPU's Instruction Set Architecture (ISA). (10, 11) However, while in Register Renaming the mapping is from a number of architectural registers to a LARGER number of physical registers (few-to-many), in VC Renaming the reverse is true: mapping is performed from a number of Virtual Virtual Channels – required by system semantics – to a SMALLER number of Physical Virtual Channels (many-to-few).

The notion of ***virtualizing virtual channels*** is an innovative idea, which, to the best of our knowledge, **has not been suggested in the literature before**. Our simulation **results are very promising**, and **hardware synthesis using commercial 65 nm TSMC libraries** demonstrates modest area/power overhead and **no impact on the router's critical path**.

The rest of this thesis is structured as follows. Chapter 2 begins with an introduction of the Network-On-Chip premise and the architecture of a generic Networks-on-Chip Router. It follows with an analysis of relative work on Networks-on-Chip Buffers, fault tolerant architectures and reconfigurable Networks-on-Chips and then describes the Networks-on-Chip Cycle Accurate Simulator we modified to perform our simulations. In Chapter 3 VC Renamer is thoroughly explained and the high-level architectures of the two VC Renamer implementations is presented. Chapters 4 and 5 analyze the high-level implementations of the Mask-Based and Linked-List-Based implementations of VC Renamer in the high-level simulator as well as the hardware implementations in an HDL language. In Chapter 6 the simulation framework is presented and a concise analysis of the results for fault-tolerant and upgradability scenarios are depicted. Chapter 7 concludes this thesis and offers a brief description for the future work on VC Renamer.

2

Networks-on-Chip - Related Work

The multi-core era is dominating the processor domain with quad-core processors available even to the general consumer. As the size of the cores increases, the importance of the interconnection backbone is stressed and needs to find ways in order to be efficient in terms of flexibility, scalability and performance. The packet-based Network-on-Chip (NoC) interconnect paradigm appears the wisest choice in terms of flexibility, scalability and performance. This chapters offers a concise introduction on Networks-on-Chip, based on the work done in (1): its vital components, basic functionality and important parameters which define it. It continues with relative work on other NoC buffering implementations, fault-tolerant architectures and re-configurable NoCs. A brief overview of the high-level simulator we modified concludes the chapter.

2.1 Networks-on-Chip

The structure of a Network-on-Chip (NoC) is similar to that of a traditional network. A communicating node interfaces to a router and each router communicates with neighboring routers, thus networking the entire chip. In the case of NoCs the communicating nodes are IP cores which are also known as Processing Elements (PEs). Supported processing elements can be anything from homogeneous or heterogeneous processors, off-chip memories, GPUs, DSPs etc. Unlike bus designs, where the communication medium may only be accessed by one sender at a time, the underlying structure of an NoC permits multiple data transfers to occur concurrently between the different on-chip routers. Data within the network is transmitted in packets. Data is disassembled into packets and packets are reassembled into data at the Network Interface (NI) of the processing element. A packet is made up of header and data fields. The header contains addressing and control information similar to traditional networks, and the data is the actual data that would be transmitted in traditional bus architectures.

The basic components of an on-chip network are its routers, the processing elements, the network interface and the physical links, as shown in figure 2.1. The figure

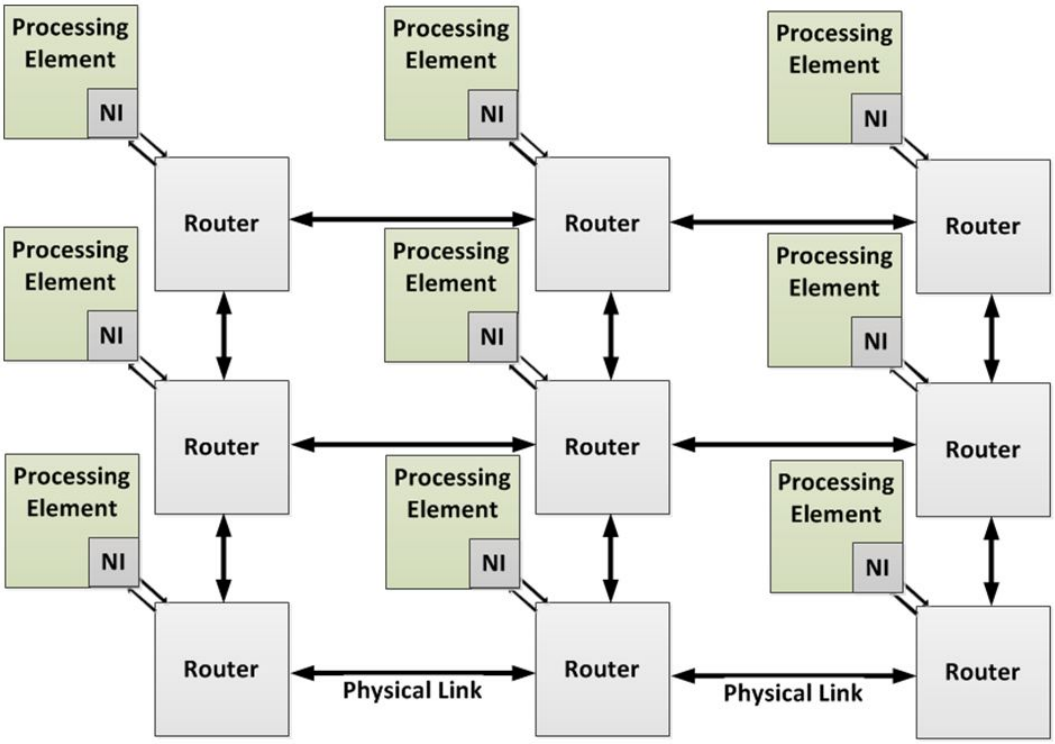


Figure 2.1: High-level overview of an on-chip network - The three major components of an on-chip network: the processing elements, the routers and the physical links

shows a partitioned chip into a 3 3 array of 9 tiles. Each tile is comprised of a PE (IP core), its network interface and the interconnected router. Each tile communicates with others via the underlying network structure. Each tile has an input port to insert packets into the network and an output port to receive packets from the network. Data transfer is done via routers, attached to each tile. The router interfaces to the rest of the network via 5 ports the PE port, and the 4 networking ports, north, south, east and west. The architecture of an NoC router is kept very simple in order to guarantee that the overall network area remains as small as possible to allow more area for the actual computational cores, the Processing Elements. A typical router consists of a crossbar, a routing decision unit, and its buffers. A router transfers data in flits. A flit is the maximum amount of information which can be transmitted from one router to another in a single clock cycle and is defined by the bandwidth of the physical link between the interconnected routers. If a packet size is greater than the flit size its broken down into head, data and tail flits. The flits are assembled and disassembled at the PEs, thus the router logic is kept as simple as possible. When a flit enters a router via one of its ports, the router uses the header information found in the flit in order to router it to the destination output port depending on the destination PE and its relative location on the chip. In order to reduce network congestion and improve performance, the routers provide buffer space, whose size depends on the overall network architecture and application needs. An NoC architecture provides significant advantages when compared to a traditional bus architecture. It offers significant improvement over bus synchronization since routers can act as pipeline stages across long wires. Its more easily expandable and can be easily reconfigured to support different communication protocols and can more easily support fault-tolerance. Its key attribute is the separation of the communication infrastructure from the computation nodes, which enables the distributed control of the network traffic and resembles the interconnection architecture of high performance parallel computing systems, on a single chip.

2.1.1 Networks-on-Chip Components

NoC Routers

The primary component for NoC architectures is the on-chip switch/router. While in traditional networks, routers are not constrained in terms of the area they must occupy and the power they can consume, in on-chip networks this is not the case. NoC routers are limited in terms of buffering space, complexity and latency, and are designed with minimal options to facilitate application requirements. A typical router consists of the port dedicated to the PE, and all the necessary ports in order to communicate with its neighboring routers (usually an additional four ports for the four cardinal directions: north, south, east, and west). An example block diagram of a NoC router is shown in Figure 2.2. The basic components of the router are its input/output ports, the crossbar switch, the buffers and the routing unit. The input/output ports receive and send data to the neighboring routers. The buffers act as intermediate stops for data waiting to be routed, in order to minimize data loss and reduce congestion. The routing unit

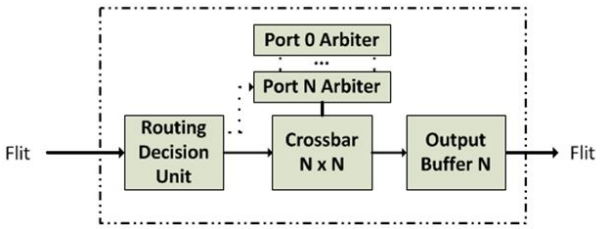


Figure 2.2: A detailed router block diagram

is responsible for directing the data between input and output ports, and finally the crossbar is the data redirection unit. A router can be pipelined or non-pipelined. A pipelined router can be clocked faster and possibly include stages for error detection and correction, priority routing, etc. A non-pipelined router forwards received data within one clock cycle, and requires minimal resources.

Input/Output Interface

The network communicates with the external world using dedicated I/O nodes, also known as ingress and egress nodes. These nodes can be either PEs or independent nodes. The ingress node(s) receive data to be transmitted to PEs from the external pins, while the egress node(s) collect data received from PEs to send outside the network. Each ingress or egress node connects to a network router. Since routers accept data in packets, ingress nodes are responsible for packetizing the data in accordance with the network packet protocol and forward the data to the network when the network is ready to receive the data. In the same manner egress nodes de-packetize the data before sending it outside the network. Ingress nodes prevent network congestion by buffering any input data that is not ready to enter the network and egress nodes provide buffering capability for data leaving the network, but cannot be accepted by the external world, thus maintaining the input-output flow without loss of data.

Processing Elements

An NoC is tasked with the objective of connecting multiple Processing Elements (PEs) that can be either homogeneous or heterogeneous. The nature of the PE is defined by the application mapped on the chip and can be any component that performs the required computations. The PE receives data from other PE(s) via the network infrastructure, performs its assigned computation task and returns the data to the appropriate PE(s) or the network output. The PEs interface to the network via specific network interface logic built in each PE. Based on the network communication protocol, each PE contains hardware to process arriving network data, by stripping off the packet header, identifying the data to be used for computation, and processing that data. When a PE wishes to send data to another PE, it uses hardware similarly designed based on the network communication protocol to packetize the output data and send it to the

network. Based on the network congestion, a PE can only send data when the network is ready to accept it.

2.1.2 Communication Overview

Data Transfer

The communication of the processing elements occurs by using the switching activity of the network infrastructure. Only data and control packets are able to travel within the network. A packet consists of two fields, header and data. The header is used to identify the data that arrives at a destination and can contain information fields such as control information, priority, sequence number, source and destination address, error correction/detection fields, etc. Just like traditional networks, header bits are used by the router in the routing procedure, used by the recipient PEs for data identification and classification, and used by the ingress/egress nodes to identify input/output data. As we have mentioned before the smallest transmissible unit that can travel between two routers is called a flit. A packet can consist of multiple flits, or a packet can be itself a flit. Breaking down packets into flits allows for smaller buffer sizes and lower wiring requirements. On the other side, breaking a packet into multiple flits increases the latency of transmission, and can result in increased congestion. When a packet is made up of multiple flits, the packet can be routed in a variety of ways, including packet switching and wormhole switching. The header information is appended on the first flit allowing the receiving router to begin computing the next destination early. The subsequent flits follow the path discovered by the header flit, ensuring that they all arrive in the same order and with no intermediate flits from another packet.

Routing

Data transmission can occur at every routing node, where each node is responsible for receiving a flit/packet and sending that to a destination node. Data transfer from node to node can happen using one of the following switching methods: store-and-forward, virtual cut-through and wormhole routing. **Store-and-forward switching:** During store-and-forward switching a packet is received and stored in its entirety before it is forwarded to the next network destination. Thus in order for a packet to be able to begin being transmitted to the next router, the last flit of each packet must be received. The router has to provide sufficient buffering capacity for each packet, and the method implies latency of at least the time required to receive the entire packet, times the number of routers the packet travels. This can be too costly when dealing with multi-hop paths. This method of routing is useful in networks where each packet is checked for integrity at every router, as it allows a single error coding to protect and repair an entire packet. **Virtual cut-through switching:** In virtual cut-through switching a packet begins to be forwarded as soon as the destination router/PE can guarantee that the entire packet will be accepted by the destination router. If the destination (either a router or a PE) cannot guarantee full reception of the packet in its entirety, then the packet is stored in the current router as a whole until the destination node is ready

to accept it. The virtual cut-through algorithm is similar to the store-and-forward method in terms of buffering requirements, however allows a faster communication as a packet can immediately begin being transmitted, provided the destination can receive it. Contrary to store-and-forward routing, the network latency depends on the congestion levels (and hence buffer utilization) in the network. **Wormhole switching:** In wormhole switching packets are split into flits and allowed to travel forward, even if sufficient buffer space for the entire packet is not available. Flits are routed as soon as the destination router/PE can guarantee acceptance of even a single flit, even when not enough buffering capacity exists for an entire packet. The moment the first flit of a packet is sent in an output port, the output port is reserved for flits of that packet, and when the leading flit is blocked, the trailing flits can be spread over multiple routers blocking the intermediate links. As a result, while wormhole routing offers significant advantages in terms of both required buffer space and minimum latency, it is more susceptible to congestion. The latency of wormhole routing is proportional to the sum of packet size (in flits) and number of hops to the destination, rather than the product of packet size (in flits) and number of hops, as in store-and-forward packet transmission.

A major problem that NoC architectures have to deal with is deadlock when routing data. Deadlock occurs when multiple packets/flits wait for interdependent resources to be released for progressing without being able to escape, as shown with the red lines of figure 2.3. There are two types of deadlocks that can happen in NoCs, buffer and channel deadlocks. A buffer deadlock occurs when all the buffers are full in a store-and-forward network. This leads to a circular wait condition, where each node is waiting for space availability to receive the next message, but as message n cannot progress, it will never release its buffer space to message n + 1. Thus a circular wait state occurs, in which no packet advances, and no buffer space is released for any packet to advance. Of similar nature are channel deadlocks which result if all channels around a circular path in a wormhole based network are busy (each node has a single buffer used for both input and output).

One possible method of avoiding deadlock is to prevent circular dependencies by carefully selecting a routing algorithm in such a way that it avoids creating data cycles among routers. Another possible resolution is the implementation of extra channels per routing node, called escape channels, which do not allow circular dependencies. The presence of escape channels ensures that in a situation where a number of packets are competing for each others resources, at least one of these packets will be able to make forward progress by using one of the the escape channels, ensuring that resources are freed, and that the remainder of the packets can also progress. Escape channels are implemented either as reserved channels in the router or virtual channels.

Due to the nature of the network structure and constrained resources, an unwise choice of a routing algorithm can be catastrophic to the overall network communication and efficiency. The main objective of the routing algorithm is to provide deadlock free and low-latency for data packets from a source PE to a destination PE. Input data is multiplexed to the output ports of each router, using the control circuit implementing the routing algorithm. There are two types of routing algorithms: deterministic

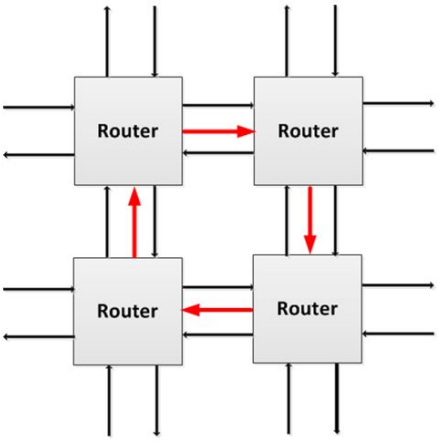


Figure 2.3: Deadlock Scenario - The red lines designate a deadlock when multiple flits wait for interdependent resources to be released for progressing without being able to escape

(determined at set-up time), or adaptive (dynamically adapting to the current network state). Deterministic routing implies knowledge of the network topology a priori, such that each router is configured with the relative location of each PE and the PEs address. Adaptive algorithms use network related information such as congestion or known faulty locations, to choose the best possible path for a packet to reach its destination.

2.1.3 Communication Structures

An NoC architecture is defined by the underlying structure of it's communication medium. Since the primary purpose of the network is to provide the most efficient communication between the PEs, the architecture that defines it is obviously one of it's most important concepts.

Network Topologies

Networks-on-chip adopted traditional network topologies due to the fact that its properties in terms of network performance are already known. Since the on-chip resources are constrained by the amount of metal layers available during design the most common topologies used are the two-dimensional mesh structure, the two-dimensional torus structure, and the three-dimensional mesh structure. The three aforementioned topologies are shown in Figure 2.4.

Of the three topologies presented in the figure the two-dimensional mesh structure is the most favoured because it's simpler to implement in terms of wiring and overall design. However the three-dimensional mesh and the two-dimensional torus structures are attractive options for specific applications due to the fact that a large number of problems map better and in a more natural form in a more highly connected topology.

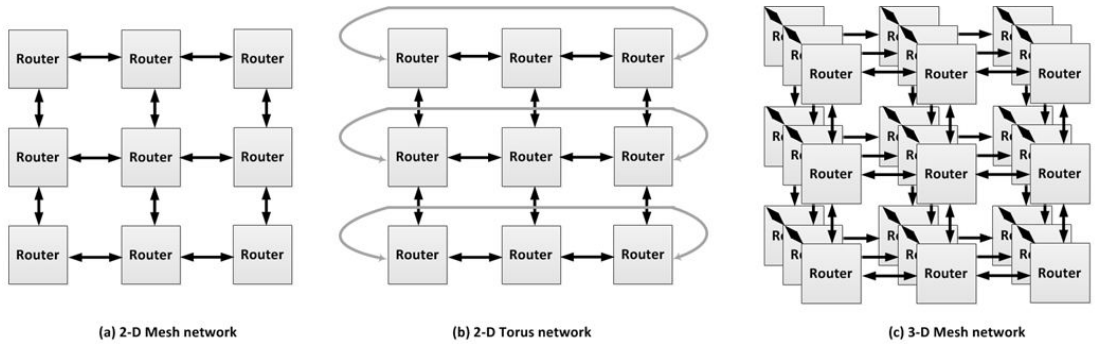


Figure 2.4: NoC Network topologies - The most common NoC Network Topologies

A particular property of the n-dimensional mesh and tori is that they do not need to have the same number of nodes in each dimension; thus the the node-to-node hop count can be reduced. This accounts to a significant benefit in terms of latency and power consumption. Toroidal structures allow packets leaving one edge of the network to enter the opposite edge of the network, significantly reducing congestion hotspots.

Data Link

The purpose of the data link is to transmit data between PEs and routers reliably, swiftly and using the least resources and power as possible. The data link consists of the physical wires that transmit data between chip components. End to end communication is achieved via the switching links (wires) which connect to the output buffers of each component. When a component is about to send data, based on the network communication protocol, the connected buffer places the data on the bit lines which transmit the data to the input buffer of the other component. When the data is received, the data link is ready to accept the next data set. Provided that the timing is correctly calculated, the receiver will read the data prior to the sender placing other data on the line. Each end point uses two ports; an input and an output. Consequently, data transmission is unidirectional, eliminating the need for a shared medium. The primary issue when communicating over such links is the handshake protocol. The handshake protocol can be implemented either synchronously, asynchronously, or using the clock and additional control/latching signals.

Crossbar

Crossbars have an N number of inputs and an M number of outputs (where M can be equal to N), and connect inputs to outputs based on the routing decision unit. The inputs and the outputs are connected via a tristate buffer. The control signal for the crossbar is generated from the routing control hardware. Typically, NoC crossbars are 5 5 (with 4 ports for input/output to the network and a port for the PE as shown in the example in Figure 2.5), but it all depends on the number of ports each router has.

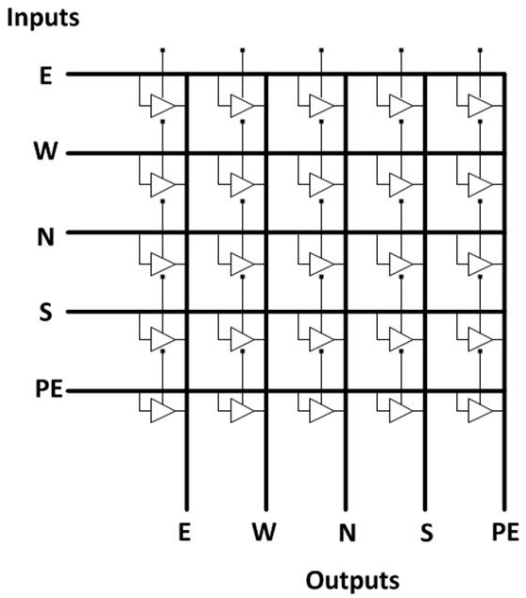


Figure 2.5: A typical NoC crossbar - The tri-state buffers are controlled by the routing hardware, and connect a given input port to the desired output port

At each connection point between a row and a column, the tristate buffer acts as the switching cell. The switching cell can either provides a connection to the row and a column or not, depending on the switching signal.

Virtual Channels

The concept of virtual channels (VCs) has been introduced in traditional networks, with the purpose of improving the QoS of the network, and to provide escape routes for deadlock scenarios. However as the NoC-concept evolved, so did the use of virtual channels which are used to enable the multiplexing of several packets onto a single physical in order to enable the use of a variety of complex routing algorithms and cache coherence protocols. The vast majority of *adaptive* routing algorithms developed for interconnection networks rely on the extensive use of Virtual Channels for deadlock avoidance and/or deadlock recovery. Similarly the plethora of cache coherence protocols designed for modern Chip Multi-Processors necessitate the use of multiple virtual networks (realized through virtual channels), in order to avoid protocol deadlocks.

Virtual channels are realized as separate data buffers where flits can be stored and each buffer's output is then multiplexed to the output resource. The operation of the multiplexer is controlled by a selection signal which determines which data to send next. It's operation is critical in ensuring QoS and efficient operation. The placement of incoming data into a particular VCs buffer is done using a number of systems, with the most common being a weighted round robin system. In such a system, channels are prioritized with larger weight values containing higher priority data than channels

with lower weight values.

2.1.4 A Virtual Channel Router

The generic architecture of a Virtual Channel NoC router is depicted in Figure 2.6. The router architecture is pipelined and comprises of five distinct stages: Routing Computation, Virtual Channel(VC) Allocation, Switch(SW) Allocation, the Crossbar and Link. During the Routing Computation Stage, the routing unit determines the next-hop direction of the header flit of a packet based on the routing algorithm employed for the on-chip network. During the Virtual Channel Allocation stage, the allocator decides which virtual channel of the receiving router (downstream router) will accept the packet based its VC availability. VC Allocation is performed into two distinct stages, VA1 and VA2. During VA1 all the packets requesting a virtual channel select one from the available pool of VCs based on the VC availability of the router's neighbours. During VA2 any arising conflicts are resolved with only one packet winning over a conflicting VC. Routing computation and VC Allocation occur only for the header flit, all subsequent flits follow the direction of the header flit. Thus a VC is occupied until a whole packet (head flit - data flits - tail flit) depart to the downstream router. During the Switch Allocation Stage a flit is selected to traverse the crossbar. A flit can only depart if the downstream router has available buffer slots within the virtual channel selected during VC Allocation. Just like VC Allocation, SW Allocation is performed in two distinct stages, SW1 and SW2. In SW1 every packet which has acquired a VC during VC Allocation selects one output port in order to traverse the link in the next clock cycle. During SW2 any arising conflicts are resolved with only one packet winning over each output port. The Crossbar stage follows the next cycle where all the packets which won the output ports during SW Arbitration put allow their flits to depart and during the Link Stage the flits traverse the link to the downstream router. In the architecture of the generic NoC architecture depicted in the figure buffer availability information is sent to the neighbouring routers using ON-OFF credits. That is in every clock cycle if a virtual channel can accommodate a flit on the next cycle it sends an ON credit signal otherwise it sends an OFF signal.

Importance of Fault-Tolerance in VC Routers

All types of faults (permanent, intermittent, and transient) may occur anywhere on the chip and may affect any system component. Defects in various locations within the Networks-on-Chip may affect the functionality of virtual channels directly (e.g., buffer faults), or indirectly (e.g., arbiter faults). The Network-on-Chip virtual channels are so deeply intertwined with the operation of the Chip Multi-Processors, that a malfunction in any of the virtual channel components may lead to network and/or cache coherence protocol deadlocks (i.e., whole-system inoperability). A network protocol deadlock appears in figure 2.7. The figure depicts an on-chip network with one VC per router port, using the deterministic XY routing algorithm. In the figure the processing element of Router 0,2 wants to send a message to Router 2,0. As the packet's flit traverse the

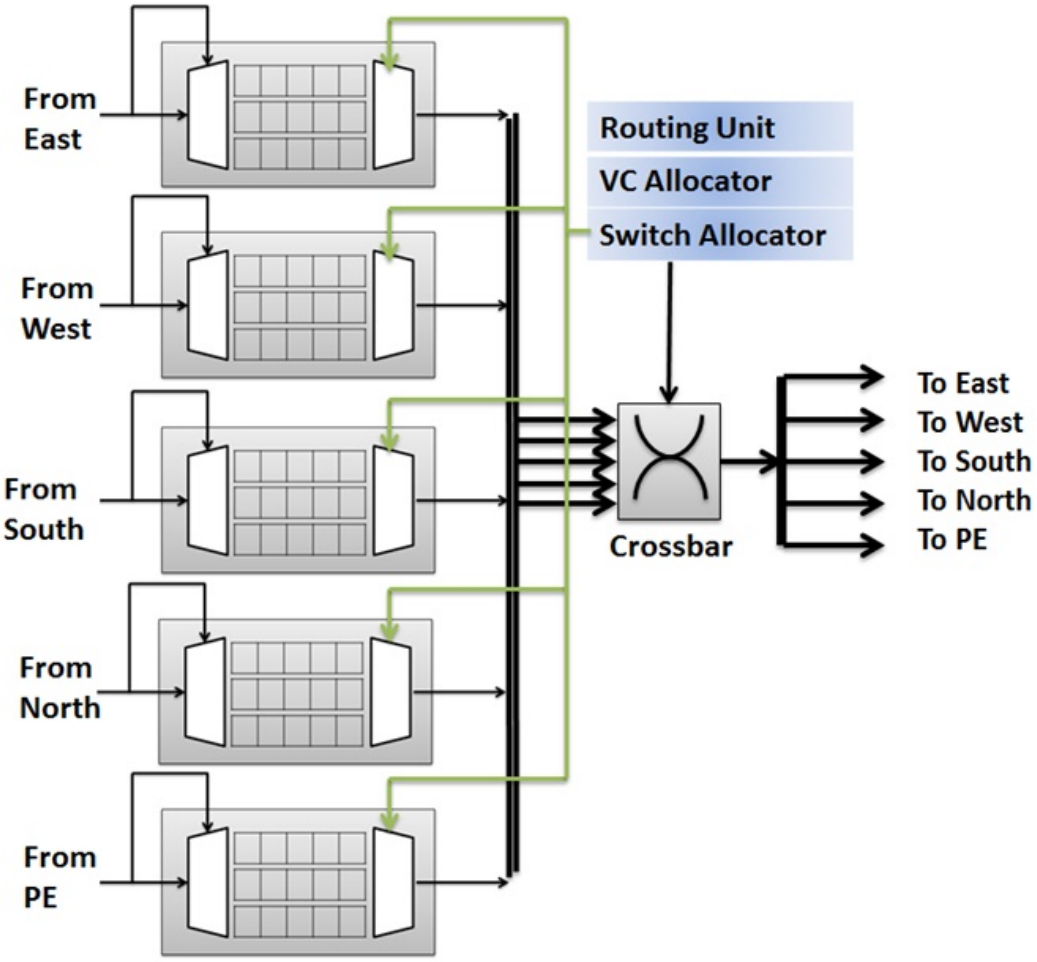


Figure 2.6: Generic Architecture of a Virtual Channel Router - The generic architecture of a virtual-channel router comprising of its 5 basic pipeline stages: Routing Computation, VC Arbitration, Switch Arbitration, Crossbar and Link Traversal

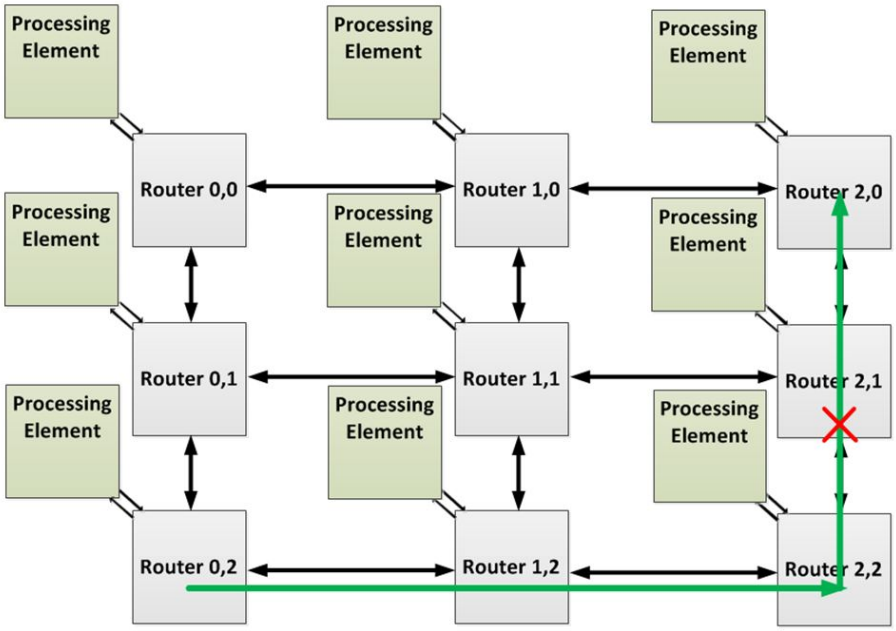


Figure 2.7: Virtual Channel Fault - Under XY deterministic routing with one VC a VC malfunction renders the NoC inoperable

network infrastructure from router to router, the South port of Router 2,1 breaks down, before they manage to cross over to Router 2,0. The flits are dropped and since no fault-tolerant policy (routing or a HW-based resolution) is used the network eventually breaks down as the flits remain clogged in the previous routers, deadlocking everything. A cache coherence protocol deadlock appears in figure 2.8. The figure depicts an on-chip network with two VCs per router port, using the deterministic XY routing algorithm. The network uses VC0 to transfer cache request messages and VC1 to transfer cache reply messages. In the figure the processing element of Router 0,2 sends a cache request message for data to Router 2,0. The cache request arrives correctly and Router 2,0 packetizes the requested data and sends it in a reply message on VC1. However east port of VC1 is broken on Router 0,0. The cache reply is dropped and the network eventually breaks down as the flits remain clogged in the previous routers, deadlocking everything.

2.2 Related Work

2.2.1 NoC Buffers

A lot of work has been done into buffering implementations of the on-chip routers and it has often been questioned whether on-chip networks ought to abandon buffering structures within the router completely in order to minimize router area and improve performance. The authors of (12, 13, 14) present implementations where an on-chip

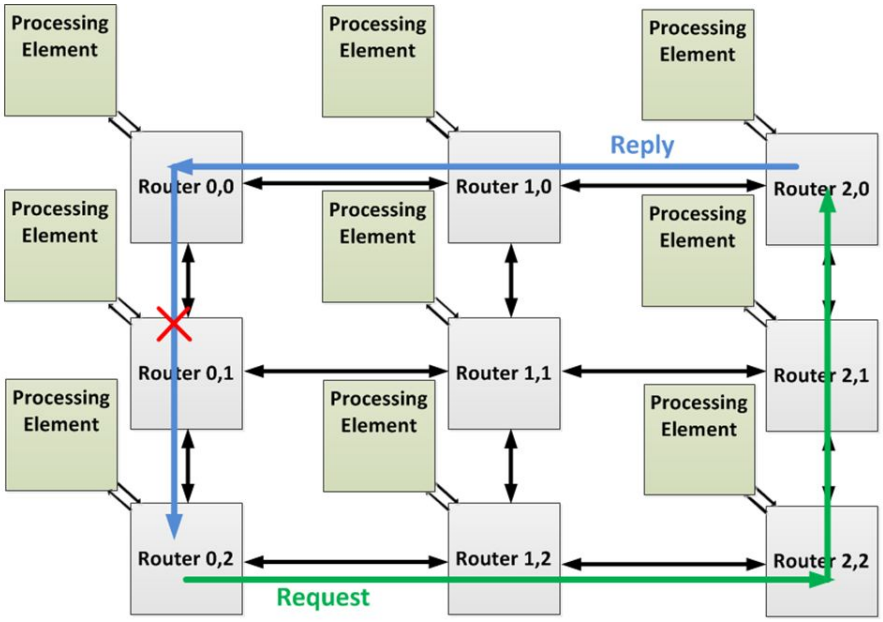


Figure 2.8: Virtual Channel Fault - Under XY deterministic routing with two VCs in a cache-coherence protocol where VC0 is used for requests and VC1 for replies a VC malfunction renders the NoC inoperable

network can be built without using buffers within the interconnecting routers. It appears though that the original design simplicity of buffer-less designs does not really lower the complexity of the overall system as it is shown in (15) where a comparison between the two has been made. The authors have reached the conclusion that the performance, cost and complexity penalties of a bufferless implementation are not really worth the removal of the router buffers.

The input buffers are an essential component of each router which aids in creating virtual channels (VCs) in order to support different flow-control protocols and various routing algorithms. It has been questioned whether using multiple physical channels might be a better option than multiplexing a number of VCs onto one physical channel, in an attempt to lower both the interconnect latency and router design complexity. The authors of (16) have established that virtual channel on-chip networks can better accommodate irregular traffic patterns which make them the wisest choice to better address the versatile needs of the many-core era with the heterogeneity of processing elements and the different application needs.

A lot of VC designs have appeared which allocate VCs statically (17, 18) or dynamically (16, 19, 20, 21, 22), using atomic VCs or centralized buffering structures (22, 23) and all of them aim to increase the performance of a VC router by reducing latency, increasing throughput at the expense of power and area overheads.

2.2.2 Fault Tolerance

In the recent years reliability has become an integral part of the design process because the future of large-scale systems depends on the ability to tolerate faults and function correctly in their presence. With each technologic scaling the shrinkage of transistors makes the components of any system susceptible to faults due to electromigration, soft errors and manufacturing defects. The importance of fault-tolerance in on-chip networks cannot be stressed out more because an error within the links or any part of the interconnecting router can render the whole network inoperable.

Faults in an on-chip network fall into two main categories, inter-router faults and intra-router faults. Inter-router faults target the links interconnecting the various routers which can either be fully or partially faulty. Resilience on the links themselves has put its weigh primarily on routing algorithms (24, 25, 26, 27) which aim to either route around them, isolating the faulty areas or send multiple copies of a message towards the destination across different routes to achieve the desired reliability. There are methodologies which target only transient faults and do not bother to discover the location or the reason of the error occurring but rely completely on error detection and error correction codes (26, 28, 29) to solve the unwanted disrupture. Such methods can only correct single-bit errors and rely on retransmission of data when multiple-bit errors are detected. (30).

Our work is mainly concerned with intra-router faults, faults which might occur within the router components. The two basic components of an NoC router are the datapath and the control logic. The datapath is comprised of the in-out ports of the router, the buffers and the crossbar. The control logic consists of the routing computation unit, the virtual channel allocators, and the switch/crossbar arbiters. A detailed analysis of the faults which might arise on an NoC network can be found in (31). We are mainly concerned with restoring normal functionality in an NoC router when an error occurs which may deem a specific virtual channel inaccessible. A read/write logical error, a crossbar error, a VC Arbiter error or even a problem in the actual VC slot may prohibit access to the VC.

The authors of (32) have tangled to create a reliable on-chip network with error detection mechanisms such as invariance checks and cyclic redundancy code checks on packets and error diagnosis which can determine the location using built-inself- test mechanisms. In case of an error they can disable faulty components and configure the functional components to work around the faulty ones while at the same time restoring the system to a previously known good state. In case of a problem in one of the ports, a port swapper is included which changes the way physical links are connected to an input port. The scenario that a whole input port might break down is unrealistic and completely disabling it can damage performance and can require adaptive routing algorithms to route around a faulty port. The authors of (33) have proposed a new router architecture which can disable a whole router in case an error arises and reallocate the extra buffers which are still functioning correctly to neighboring switches. A fault inside a specific virtual channel is more plausible, than a completely faulty router or port. VC Renamer can simply disable the faulty virtual channel while

mapping it virtually onto another physical virtual channel thus allowing the network to function correctly using simple deterministic routing algorithms without relying on fault-tolerant routing schemes which can increase both the area overhead and latency of the network.

2.2.3 Re-Configurable NoCs

There has been extensive research in architecting ***re-configurable NoCs***. However, most of this work has concentrated on the re-configuration of the router input ports, the physical links, and the network topology (34). Pertaining to the buffers, reconfigurable NoCs mainly re-allocate buffer space to the various ports, but do not provide any flexibility in the structure of the VCs (35). The work presented in this thesis can be integrated into existing reconfigurable NoC designs, in order to provide this extra flexibility.

More relevant to our work, ***dynamic NoC buffer managers*** aim to break the rigidity of static VC partitioning through run-time reconfiguration. Various dynamic VC management proposals unify the buffer resources into a common pool, which is then managed centrally and allocated to various VCs based on prevailing conditions (19, 22, 36, 37, 38, 39). Some designs can also vary the number of VCs dynamically at run-time (19, 22). These techniques, however, require a complete redesign of the input port architecture (i.e., they are disruptive) and incur significant area/power overhead, as a result of the complexity involved in maintaining multi-ported unified buffer structures and/or larger crossbar switches (19, 22, 38). In fact, most of these techniques were intended for *off-chip* communication (36, 37, 38, 39), i.e., *not as severely resource-constrained as on-chip designs.* Furthermore, these approaches are always-on: All packets are forced to go through the unified buffer. On the contrary, the proposed VC Renamer mechanism only requires minimal augmentations to the existing NoC input port architectures. No unified buffers are required and modifications are only made to the control logic of the conventional buffers. More importantly, the VC Renamer is disabled until it is actually needed, because the data path is not modified.

2.3 POP Net - A high-level cycle-accurate NoC Simulator

POP net is a cycle-accurate interconnection network simulator developed by Li-Shiuan Peh of Princeton University in 2000-2001. The simulator makes extensive use of the Standard Template Library (STL) which is a C++ library of container classes, algorithms, and iterators providing many basic algorithms and data structures useful in creating a network simulator. What makes it possible are the STL library containers which are highly parametrizable and can be used as templates for the various classes which will build up the NoC. We modified this simulator to implement both versions of the VC Renamer mechanism. A detailed analysis of the simulator is presented in A

3

VC Renamer High-Level Architecture

The VC Renamer mechanism is completely transparent to the neighboring NoC routers. The routers are only aware of the presence of a specific number of VVCs (those required by the routing algorithm and/or the coherence protocol). A disabled PVC in one router will not become visible to the neighbors; VC Renaming will hide the anomaly by remapping the affected VVC onto a different PVC. System functionality will be fully maintained, albeit at a degraded performance. Note that PVC atomicity is now broken, i.e., each PVC may hold more than one VVC. Atomicity within the individual VVCs, however, is still maintained.

The critical issues of *starvation* and *protocol-level deadlocks* arise whenever two or more VVCs are multiplexed over a single PVC. If one VVC dominates the available buffer space, other VVCs may not be left with any available slots, leading to protocol-level deadlocks (i.e., the higher-level protocol expects a particular message type that never arrives, due to the above-mentioned problematic situation within a PVC). The VC Renamer architecture ensures the absence of such pathologies through (a) intelligent use of the *credits* mechanism, which regulates flow between the PVCs, and (b) the assumption that the number of VVCs mapped to a specific PVC cannot exceed the number of buffer slots physically present in the PVC. The credits mechanism will be explained in Section 4.

Two different VC Renamer implementations have been developed: A **Mask-Based (MB) implementation** and a **Linked-List-Based (LLB) implementation**. Both versions offer the same functionality, but each is geared toward a different objective. The MB approach is extremely lightweight and targets fault-tolerant designs, where the VC Renamer mechanism will only be used in input ports affected by faults. The LLB approach incurs a slightly higher hardware overhead and targets system *upgradeability*, whereby the VC Renamer technique will be used in *all* routers *simultaneously*.

It should be noted here that *both* implementations **do *not* affect the VC Allocation (VA) or Switch Arbitration (SA) pipeline stages of the router**. Since only *one* of the VVCs mapped to a particular PVC is active in any given clock cycle,

the VA and SA arbiters are completely unaffected. This attribute is of paramount importance, since the VA and SA stages usually determine the router's critical path. It will be demonstrated through hardware synthesis that **the VC Renamer operates within the slack of the other pipeline stages and does *not* impact the router's critical path**. Moreover, as the VA and SA arbiters are unaffected, **any arbiter prioritization policies can still be used**.

3.1 Mask-Based High-Level Architecture

In a typical NoC router input port, each PVC is realized using a k-deep FIFO buffer, where k is the maximum number of flits (a *packet* comprises a number of fixed-size *flits*) in a PVC, as shown in Figure 3.1. The figure presents a high-level overview of the MB architecture. FIFO order within each PVC is maintained by *head* and *tail* pointers. These pointers are k-deep, 1-wide, 1-hot circular registers. Assuming that the head of the buffer is on the right-hand side (see Figure 3.1), the pointers move *from right to left*. Upon arrival of a flit, the *tail* pointer advances one position to the left; upon departure of a flit, the *head* pointer advances one position to the left. In the MB implementation of the VC Renamer, each PVC maintains its generic head and tail pointers. Two main additions are made: (1) a *VVC-to-PVC Mapping Table*, and (2) a k-bit *Mask* for each supported VVC, which indicates the occupied positions in the respective slots of the PVC. The ordering of the flits within each VVC is kept by only allowing new flits to be stored in available PVC slots that are located *to the left* of the left-most '1' in the VVC mask (remember, flits are assumed to fill in a VC buffer *from right to left*). Thus, a particular VVC builds its mask from right to left – as flits arrive – and tears down its mask from right to left– as flits depart.

The ***arrival*** of flits is regulated by the credits mechanism. Credits are only sent to the upstream router (i.e., the potential sender) if a preliminary *arrival test* is successful. Note that the proposed VC Renamer design assumes the use of *on/off credits*, i.e., *stop/go* signals (for each VVC) that regulate flow based on PVC buffer availability. Credits are sent to upstream routers for each VVC in each input port. The credits signals for all VVCs mapped to a specific PVC are determined by the slot availability in said PVC; i.e., even though credits are distributed at the VVC-level, they are determined by buffer availability at the PVC-level. To ensure that the VVCs mapped to a particular PVC do *not* receive credits *simultaneously*, only credits for one VVC (per PVC) are dispatched in any given clock cycle. Without loss of generality, **the credits for each VVC *mapped to a single PVC* are sent out in round-robin fashion (one in each cycle)**. As will be shown later on, **this round-robin dispatch of credits has a minimal impact on performance**. If needed, the round-robin policy can be replaced by any other policy, in order to implement different VVC prioritization schemes.

The Arrival Test algorithm – which is responsible for the credits mechanism – is shown in **Algorithm 1**. Step 1 of the algorithm ensures the absence of *starvation* and *protocol-level deadlocks*. Specifically, the two conditions of Step 1 guarantee that

each VVC mapped to a specific PVC will always have access to at least one buffer slot. Remember that the number of VVCs mapped to a specific PVC cannot exceed the number of buffer slots physically present in the PVC. An "Empty" VVC is one whose Mask contains all zeros. The conditions of Step 1 are independent and can be fully parallelized in hardware. In addition to this check (Step 1), a flit is only allowed to arrive if the position pointed to by the PVC tail pointer is to the left of the left-most '1' in the corresponding VVC mask (remember, flits are assumed to fill in a VC buffer *from right to left*, as shown in Figure 3.1). This ensures that the flits of a particular VVC remain in order within the PVC buffer. This critical condition is checked through the *subtraction* of the VVC mask from the 1-hot PVC tail pointer (Step 2 of Algorithm 1) and observing the *sign* of the result (the *left-most bit* is assumed to be the *most significant bit*). Credits are sent to the upstream router if the PVC tail pointer is greater than the value of the VVC mask (positive subtraction result) *and* the PVC tail pointer does not point to an occupied position (Step 3). Since flit departures from the various VVCs may leave the PVC buffer fragmented, the PVC tail pointer may point to an occupied position. In such a case, no credits are sent out in the current cycle and the PVC tail pointer is shifted to the left. When a flit arrives at an input port, the VVC ID (contained within the flit) is used to index into the *VVC-to-PVC Mapping Table*, which uses the corresponding PVC ID to de-multiplex the incoming flit to the appropriate PVC. The existing PVC tail pointer is used to store the flit into the buffer. At the same time, the position of the PVC tail pointer denotes the respective bit position in the active VVC mask that must be set to '1'.

An example of how the *arrival test* functions is illustrated in Figure 3.2. In order for a flit to be able to arrive from the neighbouring router, a credit ON signal must first be sent. In the said figure we perform the steps from Algorithm 1 for VVC1. We first have to examine if there are available slots in the PVC buffer. Since both mapped VVCs have occupied slots in the PVC k=0. We have four available slots so the first condition of the algorithm is true. During the second condition test we examine whether the value of Tail Pointer (00010000) is greater than that of the VVC1's Mask (00000110). The second condition also holds. The last condition is to examine whether the tail pointer points to a free slot. Since the tail pointer does point to a free slot all the conditions hold and we sent an ON credit signal for VVC1. When an incoming flit for VVC1 arrives, it's stored at the position of the tail pointer and the VVC1 Mask bit is set for that postion after which the tail pointer advances by one position. Note that if we were to send a credit signal for VVC0 the tail-mask comparison wouldn't hold (00010000 ¡ 00101000) and we would have sent a credit OFF signal for VVC0.

Algorithm 1 Mask-Based Mechanism - Arrival Test

k = # of Empty VVCs mapped to current PVC
1: If {(PVC_Free_Slots > k) OR (Current_VVC == Empty)} AND
2: (Tail Pointer > VVC Mask Value) AND
3: (Tail Pointer points to Free Slot) THEN
4: => VVC_X Credits = ON

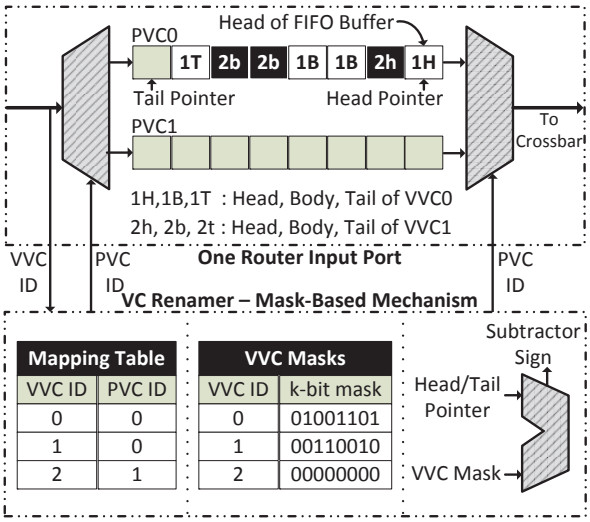


Figure 3.1: High-level overview of the Mask-Based (MB) implementation of the VC Renamer. Only one input port is shown for clarity. In this example, VVC0 and VVC1 are mapped to PVC0, while VVC2 is mapped to PVC1. Note that one subtractor *per PVC* is required.

Algorithm 2 Mask-Based Mechanism - Departure Test

1: If (Downstream_Router_Credits = ON) AND
2: (Head Pointer points to Occupied Slot) AND
3: (Head Pointer > VVC Mask Value) THEN
4: => Allow Flit to Depart

In a similar manner, a flit is only allowed to ***depart*** if the position pointed by the PVC head pointer is the position of the right-most '1' in the corresponding VVC mask. This ensures that the flits of a particular VVC always depart the PVC buffer in the correct order. The *departure test* (as shown in **Algorithm 2**) is performed by doing a simple subtraction of the VVC mask from the 1-hot PVC head pointer (Step 3) and observing the *sign* of the result (the *right-most bit* is assumed to be the *most significant bit*). In the subtraction, the bit position in the VVC mask where the PVC head pointer points to is set to '0'. The flit is allowed to depart if the PVC head pointer does not point to an empty position (Step 2) *and* the PVC head pointer is greater than the value of the VVC mask (Step 3, positive subtraction result). If the PVC head pointer points to an empty position, no flit departs the buffer and the head pointer is shifted to the left. If the flit of one VVC cannot depart (e.g., no space in the downstream router), the PVC head pointer moves to the next position in the following clock cycle, in order to allow other VVCs to proceed and avoid Head-of-Line (HoL) blocking.

An example of how the *departure test* functions is illustrated in Figure 3.2. A flit is allowed to depart from the port if the three conditions from Algorithm 2 hold. In the said figure the head pointer points to a VVC1 flit so we only need to perform the

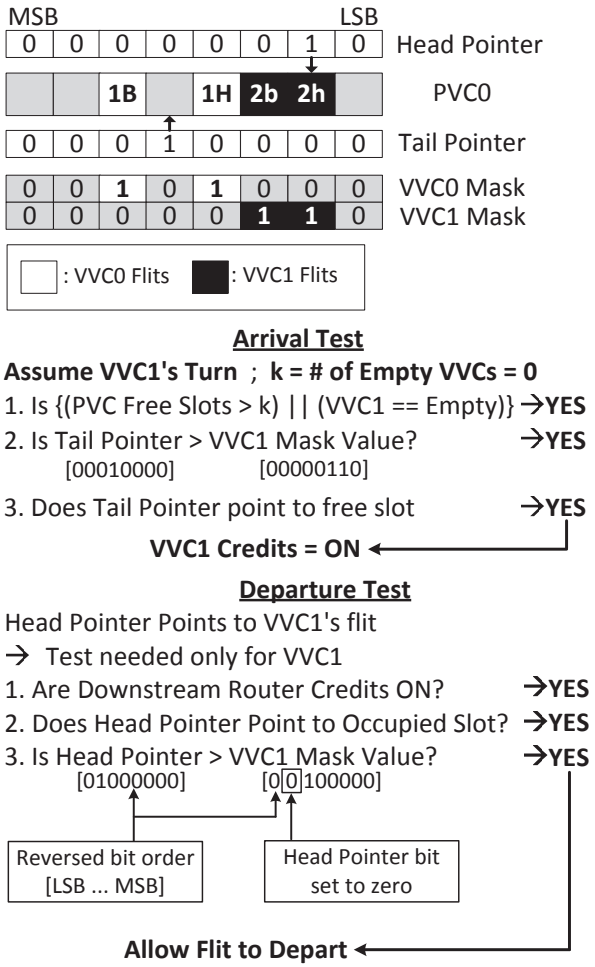


Figure 3.2: Step-by-step examples of the *arrival* and *departure tests* of the Mask-Based (MB) implementation of the VC Renamer. Only one PVC is shown for clarity with two VVCs (VVC0 and VVC1) mapped onto it. The various steps correspond to Algorithms 1 and 2.

test for VVC1. We first have to examine that the downstream router has sent a Credit ON signal and that the head pointer does indeed point to an occuppied position. Since both those conditions hold we must then verify that the Head Pointer value is greater than the value of the VVC1 Mask. Since the mask is build from right to left it must be teared down from right to left. The final comparison achieves just that but in order to do so the values of both the Head Pointer and the Mask Value must be inverted and the mask bit of the head pointer position must be reset. In our case the value of the head pointer (01000000) is greater than the value of VVC1's mask (00100000) so since all three conditions hold, the flit is allowed to depart. When a flit departs the mask bit where the head pointer points is reset and the head pointer advances by one position.

The steps in both Algorithms 1 and 2 can be *fully parallelized and overlapped in*

hardware, thus incurring minimal latency overhead. More importantly, they are *off* the router's critical path (which lies in the VA/SA stages).

Note that credits are sent to upstream routers for each VVC in each input port. To ensure that the VVCs mapped to a particular PVC do *not* receive credits *simultaneously*, only credits for one VVC (per PVC) are dispatched in any given clock cycle. Without loss of generality, **the credits for each VVC *mapped to a single PVC* are sent out in round-robin fashion (one in each cycle)**. As will be shown later on, **this round-robin dispatch of credits has a minimal impact on performance**. If needed, the round-robin policy can be replaced by any other policy, in order to implement different VVC prioritization schemes.

3.2 Linked-List Based High-Level Architecture

As will be demonstrated in Section 6, the mask-based approach incurs a performance penalty. Hence, the Linked-List-Based (LLB) implementation of VC Renamer targets higher performance at the expense of a slightly higher area/power overhead, as compared to the MB implementation. As in the MB approach, the modifications required to realize the LLB mechanism only affect the *control logic* of the existing PVC buffers. The new components comprise: (1) a *PVC Pointer List*, (2) a *Free-Slot FIFO List*, (3) a *Front-of-VC List*, (4) a *Back-of-VC List*, and (5) the same *VVC-to-PVC Mapping Table* of the MB implementation. Figure 3.3 shows a high-level overview of the LLB architecture.

Assuming k-deep PVC buffers, the *PVC Pointer List* contains one k-deep, $\log_2 k$-bit wide vector per PVC. This vector holds the pointers to the next flit of each packet. The first flit of each VVC mapped to a specific PVC is located by accessing the *Front-of-VC List*, which is a list containing the location ($\log_2 k$ bits) of the next-departing flit of each VVC. The *Front-of-VC List* replaces the *PVC head pointer*. Once the location of the next-departing flit is known, the *PVC Pointer List* points to the subsequent flits of the same packet in a linked-list manner. Similarly, the *Back-of-VC List* contains the location ($\log_2 k$ bits) of the last-stored flit of each VVC. It is used to extend the linked-list (i.e., add a new pointer) in the *PVC Pointer List* whenever a new flit of an in-flight packet arrives. The *Free-Slot FIFO List* maintains the free slots in each PVC. There is one such k-deep, $\log_2 k$-bit wide FIFO structure for each PVC in the input port. The *Free-Slot FIFO List* supplies the write locations for new incoming flits. (i.e., it replaces the *PVC tail pointer* and is used to update the PVC Pointer List (with a new tail pointer) and the Back-of-VC List upon a flit arrival.)

The credits are sent to upstream routers in the same round-robin manner as in the MB approach (see Section 3.1). Unlike the MB technique, the credits in the LLB implementation are regulated only by the two conditions shown in Step 1 of Algorithm 1, i.e.,

$$\text{If } \{(\text{PVC\_Free\_Slots} > k) \text{ —— } (\text{Current\_VVC} == \text{Empty})\}$$
$$=> \text{VVC\_X Credits} = \text{ON},$$
$$\text{where } k = \#\text{ of Empty VVCs mapped to current PVC.}$$

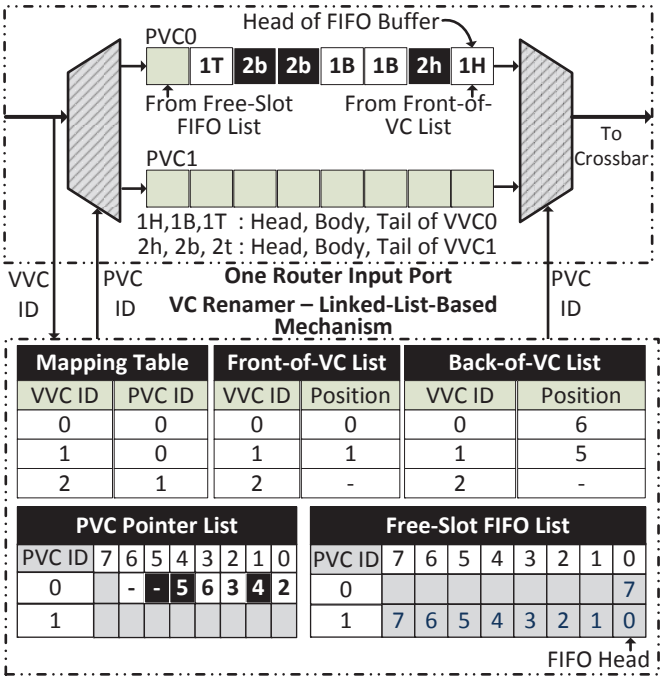


Figure 3.3: High-level overview of the Linked-List-Based (LLB) implementation of the VC Renamer. Only one input port is shown for clarity. In this example, VVC0 and VVC1 are mapped to PVC0, while VVC2 is mapped to PVC1.

As previously mentioned, this check ensures the absence of starvation and protocol-level deadlocks. In this case, the number of free slots is determined by the occupancy of the *Free-Slot FIFO List*, and the number of "Empty" VVCs is determined by the number of invalid entries in the *Front-of-VC List*.

Algorithm 3 Linked-List-Based Mechanism - Flit Arrival

1: Get empty slot from *Free-Slot FIFO List*
2: Store flit in PVC
3: **If** (VVC = empty) update *Front-of-VC List* with flit position from *Free-Slot FIFO List*
4: **Else** use *Back-of-VC List* to index into the *PVC Pointer List* and extend the linked-list
5: Update *Back-of-VC List* with new flit position

When a new flit ***arrives*** at the input port (see **Algorithm 3**), the VVC ID within the flit is used to acquire the corresponding PVC ID from the *VVC-to-PVC Mapping Table*. The *Free-Slot FIFO List* is used to point the new flit to an available PVC slot (Steps 1 and 2). If the *Front-of-VC List* entry for the particular VVC is empty (i.e., start of new packet), it is updated with the location granted from the *Free-Slot FIFO List* (Step 3). At the same time, the corresponding *Back-of-VC List* entry is updated

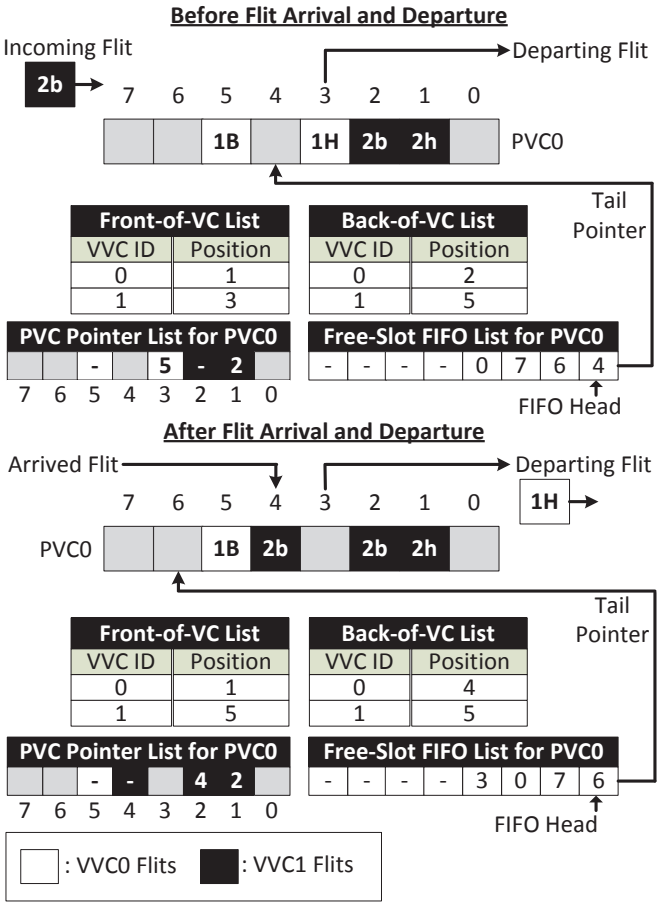
with the new location (Step 5). If the *Front-of-VC List* entry for the current VVC is *not* empty (i.e., the new flit is part of an existing packet), then the *Back-of-VC List* is used to index into the *PVC Pointer List* in order to extend the linked-list to the PVC location of the new flit (Step 4).

Algorithm 4 Linked-List-Based Mechanism - Flit Departure

1: Acquire new Head Pointer from the *Front-of-VC List*
2: Allow flit to depart if credits are available
3: Push the head pointer value to the *Free-Slot FIFO List*
4: **If** (flit = tail flit) invalidate *Front-of-VC List* and *Back-of-VC List* entries
5: **Else** use the Head Pointer value to index into the *PVC Pointer List* and update *Front-of-VC List* with acquired value

Flit ***departure*** follows a similar process (see **Algorithm 4**). Once a flit is selected to depart, its VVC ID is used to acquire the next-departing flit location from the *Front-of-VC List* (Step 1). This value is also enqueued within the *Free-Slot FIFO List*, since the location will be vacated (Step 3). If the departing flit is a *tail* flit (i.e., the end of a packet), the corresponding entries in the *Front-of-VC List* and the *Back-of-VC List* are invalidated (Step 4, indicating an empty VVC). If the departing flit is *not* the last flit of its packet, then the *Front-of-VC List* is used to index into the *PVC Pointer List*. The indexed value within the *PVC Pointer List* points to the PVC location of the *next* flit of the same packet. This value is used to update the *Front-of-VC List*; i.e., the location of the *next* flit of the same packet now becomes the new head of the VVC (Step 5).

An example of how the *arrival* and *departure mechanisms* function is illustrated in Figure 3.4. The said figure shows an arriving flit for VVC1 and a departing flit for VVC0. When the flit arrives (Algorithm 3) we first get an empty slot position from Free-Slot FIFO List which is used as the tail pointer (Step 1), in this case position 4. We then store the flit in position 4 (Step 2). Since VVC1 was not empty when the flit arrived there is no need to update the Front-of-VC List (Step 3). Afterwards we need to update the PVC Pointer list and we can do so by storing the value of the tail pointer (Position 4) inside the PVC Pointer List at the position where the Back-of-VC List for VVC1 points (Position 4) (Step 4). Finally the Back-of-VC List is updated with the position 4 of the tail pointer (Step 5). When a VVC0 flit departs (Algorithm 4 we first have to acquire the head pointer value from the Front-of-VC List for VVC0 (Step 1). In this case the head pointer attains the position 3 value and the flit is allowed to depart (Step 2). Afterwards we push position 3 to tail of Free-Slot FIFO List (Step 3). Since the departing flit is not the last flit of that VVC there's no need to invalidate either the Front-of-VC nor the Back-of-VC Lists (Step 4). Finally the value of the head pointer (Position 3) is used to index the PVC Pointer List whose value (Position 5) is stored in the Front-of-VC List for VVC0 (Step 5). Again, the steps in both Algorithms 3 and 4 can be *fully parallelized and overlapped in hardware*, and they are *off* the router's critical path. The credits are sent to upstream routers in the same round-robin manner



Flit Arrival – Incoming flit belongs to VVC1

1: Acquire Tail Pointer from Free-Slot FIFO List → Position 4
2: Store Flit in Position 4
3. If (VVC != Empty) → Do Nothing
4: PVC Pointer List [Back-of-VC List[VVC1] = 2] = Position 4
5: Back-of-VC List[VVC1] = Position 4

Flit Departure – Assume VVC0's turn

1: Acquire Head Pointer from Front-of-VC List[VVC0] → Position 3
2: Remove flit from PVC Position 3
3: Push Position 3 to tail of Free-Slot FIFO List
4: If (Flit != Last flit) → Do Nothing
5: Front-of-VC List[VVC0] = PVC Pointer List [3] = Position 5

Figure 3.4: Step-by-step examples of the *arrival* and *departure mechanisms* of the Linked-List-Based (LLB) implementation of the VC Renamer. Only one PVC is shown for clarity with two VVCs (VVC0 and VVC1) mapped onto it. The various steps correspond to Algorithms 3 and 4.

as in the MB approach. Credits are sent out based on the availability indicated in the *Free-Slot FIFO List*.

4

VC Renamer - Mask-Based Implementation

4.1 Implementation in High-Level Simulator

We implemented the Mask-Based mechanism in the cycle-accurate on-chip network simulator implemented in C++, POPNet. The simulator is analyzed thoroughly in Section 2.3 of Chapter 2. Its important to stress out the point that the VC Renamer mechanism can be enabled on and off in accordance to the requirements of a port i.e. in case of a faulty virtual channel or in case we need to support a greater number of VCs on that port. As it is shown in Chapter 2, the POPNet router pipeline is comprised of five basic stages: Routing Decision, VC Arbitration, SW Arbitration, the flit-out buffer stage and the flit traversal stage. This section presents, through a series of flow-control diagrams which are featured in B, the basic modifications needed in the high-level simulator in order to support VC Renamer and to be able to turn-it on and off according to the needs of the network. Alongside the 5-pipeline stages modifications also had to be made to the way flits are handled when they arrive, to the way credits are sent and some basic actions which needed to be executed at the end of the pipeline.

- **Incoming Flits:** When a flit arrives at an input port, a check is made to see if VC Renamer is enabled on the selected port. If it is the flit is stored in the appropriate PVC using the VVC-to-PVC mapping table and the appropriate VVC mask bit is set using the current value of the tail pointer. If the VC Renamer bit is disabled the flit is simply stored in the regular PVC. (Figure B.1)
- **Router Pipeline Stage 1: Routing Decision:** During this stage a for loop begins examining all the router ports to see if routing needs to commence. For each of the ports if VC Renamer is disabled all the normal VCs are examined and if any of them are in the routing state (due to the arrival of a header flit), the appropriate routing function is called which returns all the port-VC pairs of the downstream router and stores them in a port-vc pair vector. If VC Renamer is enabled all the PVCs are being examined one by one. For the PVCs which

only have one VVC mapping the normal procedure occurs, if the mapped VVC is in the routing state (due to the arrival of a header flit), the appropriate routing function is called which returns all the port-VC pairs of the downstream router and stores them in a port-vc pair vector. For the PVCs which have more than one VVC mappings the head-pointer of the PVC is examined to discover to which VVC that flit belongs. if the mapped VVC for that flit is in the routing state, the appropriate routing function is called which returns all the port-VC pairs of the downstream router and stores them in a 2d vector. When all the ports are examined, the simulator moves onto the stage 1 of the VC Arbitration phase. (Figure B.2)

- **Router Pipeline Stage 2: VC Arbitration VA1 Stage:** During the VC Arbitration, VA 1 Stage a loop examines all the router ports. If VC Renamer is disabled all the VCs are examined and if a particular VC is in the VC Arbitration state the vc selection function is called which uses all the port-vc pairs from the routing decision stage to find and return an available port-VC pair which is stored in a table and used in the VA2 phase. If VC Renamer is enabled all the PVCs are being examined one by one. For the PVCs which only have one VVC mapping the normal procedure occurs, the mapped VVC is examined and if it is in the VC Arbitration state the vc selection function is called which uses all the port-vc pairs from the routing decision stage to find and return an available port-VC pair which is stored in a table and used in the VA2 phase. For the PVCs which have more than one VVC mappings the head-pointer of the PVC is examined to discover to which VVC that flit belongs. If the mapped VVC for that flit is in the VC arbitration state, the vc selection function is called which uses all the port-vc pairs from the routing decision stage to find and return an available port-VC pair which is stored in a table and used in the VA2 phase. When all the ports are examined, the simulator moves onto the stage 2 of the VC Arbitration phase. (Figure B.3)

- **Router Pipeline Stage 2: VC Arbitration VA2 Stage:** During the VA2 Stage all of the ports are being examined one by one and for each port all of the VCs (in the case where VC Renamer is disabled) and all of the VVCs (in the case where VC Renamer is enabled) are checked to see if they managed to acquire an output VC during the VA1 Stage. PopNET uses a random selection algorithm to grant a VC/VVC an output port. The algorithm basically examines if the output VC chosen by a VC/VVC is available and if it is, it's assigned to that VC/VVC and its state is updated to SW AB to be able to move to the Switch Arbitration. (Figure B.4)

- **Router Pipeline Stage 3: SW Arbitration SA1 Stage:** During the SW Arbitration SA 1 Stage a loop examines all the router ports. If VC Renamer is disabled all the VCs are examined and if a particular VC is in the SW Arbitration state, the assigned output VC is examined to see if there are available credits and if there are, the VC number is stored inside a table which is used when SA2

commences. If VC Renamer is enabled all the PVCs are being examined one by one. For the PVCs which only have one VVC mapping the normal procedure occurs. The assigned output VC, for the requesting VVC, is examined to see if there are available credits and if there are the VC number is stored inside a table which is used when SA2 commences. For the PVCs which have more than one VVC mappings the head-pointer of the PVC is examined to discover to which VVC that flit belongs. if the mapped VVC for that flit is in the SW arbitration state, the assigned output VC, for the requesting VVC, is examined to see if there are available credits and if there are the VC number is stored inside a table which is used when SA2 commences. When all the ports are examined, the simulator moves onto the stage 2 of the SW Arbitration phase. (Figure B.5)

- **Router Pipeline Stage 3: SW Arbitration SA2 Stage:** During the SW Arbitration SA 2 Stage a loop examines all the router ports and for each output port only one request can be satisfied with a random VC/VVC being selected. The winning request updates the state of the VC/VVC from SW AB to SW TR to be able to follow to the flit outbuffer stage. (Figure B.6)

- **Router Pipeline Stage 4: Flit Outbuffer Stage:** In the flit outbuffer stage the VCs/VVCs of each port are examined to see if they are in the switch traversal state. If that is the case the flit is removed from the VC and added to the appropriate output buffer. The ports are examined one by one and if VC Renamer is disabled on a port, all of its VCs are examined and if a particular VC is in the SW Traversal state, a flit is removed from the VC and added to the appropriate output buffer. If the removed flit is at its destination it is consumed by the router and if its the tail flit the VC is released. If VC Renamer is enabled on the selected port, the VVCs are checked one by one. If the selected VVC is the only one mapped onto a PVC the normal procedure is followed.If it's in the SW Traversal state, a flit is removed from the VVC and added to the appropriate output buffer. If the VVC is mapped onto a PVC which has more than one VVC mappings the head pointer is examined and if it points to a flit belonging to that VVC, the VVC state is examined. If it's in the SW traversal state, a comparison is made to see if the head pointer is greater than the VVC mask. If it is a flit from that PVC is removed and added to the correct output buffer. When all the router ports are examined the simulator proceeds to the Flit Traversal stage. (Figure B.7)

- **Router Pipeline Stage 5: Flit Traversal Stage:** During the flit traversal stage each output buffer (North, South, East, West) is examined and if its not empty, a flit us removed and sent to the appropriate downstream router. When all the output buffers for all the ports are examined a router cycle ends. (Figure B.8)

- **Credit Mechanism:** It's important to note that the generic NoC router implemented within the high-level simulator originally used counter-based credits.

In the counter-based credits scheme, each router holds a counter for each of its neighbouring VCs initialized to the number of slots of that particular VC. When a flit is sent from the current router, the value of the counter is decremented by one and when the flit departs from the router whom the current router sends a flit to, a signal informs it to increase that counter by one. In order for VC Renamer to work, an ON-OFF credit mechanism is required where in each cycle a router sends an ON signal for each of its VCs to its downstream router if it can accommodate at-most one flit and an OFF signal if it can't. An extra process is added to the pipeline of the router which is performed in parallel with the basic router functions where in every clock cycle ON-OFF credits are being sent based on the slot availability of each VC. In each port where VC Renamer is disabled, each VC is examined and if it can accommodate one flit an ON credit signal is sent, otherwise an OFF credit signal is sent. It's important to note that, as shown in the figure, the check performed is not for just one available slot but for four, in order to accommodate the four cycle turn-around time needed for a flit to arrive at the downstream router. If VC Renamer is enabled, and there's only one mapped VVC on a particular PVC the same procedure is used. For PVCs which have more than one VVC mappings, credits are sent in a round-robin fashion for each of the mapped VVCs. i.e. for three mapped VVCs it would be: VVC0-ON VVC1-OFF VVC2-OFF, VVC0-OFF VVC1-ON VVC2-OFF, VVC0-OFF VVC1-OFF VVC2-ON... For the VVC whose turn is to send an ON credit signal a comparison is performed between the tail-pointer and the VVC's mask value, if the tail-pointer is greater than that of the mask value and there are more than 5 consecutive empty buffer slots (once again to accommodate the turn-around time and avoid mask violations) an ON signal is sent. To avoid the possibility of leading the network into a state of deadlock, when there are less than five consecutive slots available a counter is used which examines if the tail-pointer is stuck. When that counter reaches 30 an ON signal is sent to allow the network to avoid deadlock. When all the ports are examined the credit function exits. (Figure B.9)

- **VC Renamer End of router pipeline Tasks:** When the router pipeline ends, all of the ports are examined one by one and for the ports where the VC Renamer mechanism is enabled, every PVC with more than one VVC mapping adheres to the following process. If the PVC is empty, the head and tail pointer are reset, as well as the stuck head and stuck tail pointer counters in order to enhance the performance of the mask-based mechanism. The tail pointer is then examined and if it points to an occupied position it advances by one position. Then the head pointer is examined and if points to an empty position it advances by one position. A final check is made to the head pointer, where if it has been stuck for more than 20 cycles it advances by one position and the stuck head pointer counter is reset. This is done once again to ensure that the network will not in any case get stuck in a state of deadlock if any of a VVC's flits are not advancing within the network. After these tasks a new router cycle begins. (Figure B.10)

Table A: Mask-Based Implementation Storage Cost

Generic: 2 8-slot 128-bit VCs	Generic	VC Renamer – Mask-Based		
		2 VVCs /PVC	3 VVCs /PVC	4 VVCs /PVC
	10240 bits	10400 bits **1.56%**	10480 bits **2.34%**	10580 bits **3.12%**

Table B: Mask-Based Implementation – Required Hardware Area Cost

Generic: 2 8-slot 128-bit VCs	VC Renamer – Mask-Based		
	2 VVCs /PVC	3 VVCs /PVC	4 VVCs /PVC
	2.09%	2.87%	5.36%

Table C: Mask-Based Implementation – Required Hardware Power Cost

Generic: 2 8-slot 128-bit VCs	VC Renamer – Mask-Based		
	2 VVCs /PVC	3 VVCs /PVC	4 VVCs /PVC
	0.23%	2.33%	3.71%

Figure 4.1: Mask-Based Storage Cost - Cost of the Mask-Based Implementation compared to a Generic NoC Router as the number of mapped VVCs increases

4.2 Implementation in HDL Language

In order to quantify the actual hardware overhead of the Mask-based Implementation in comparison with the generic NoC architecture, as well as verify that the critical path of the router remains the same we had to get actual results concerning the area, power and timing costs of our implementation. To get an initial feel of the cost of our implementation we put quantified the storage cost required for the Mask-Based approach compared to that of a Generic NoC Network.

Generic NoC Network Storage Cost

- Buffer Slots : # of ports X # of VCs X # of buffer slots X flit size

VC Renamer Mask-Based Storage Cost

- Buffer Slots : # of ports X # of VCs X # of buffer slots X flit size
- VVC Masks : # of ports X # of VVCs X # of buffer slots

It can be seen from Table A of the figure 4.1 that a generic NoC network with 2 8-slot 128-bit PVCs requires 10240 bits of storage space. If we enable VC Renamer to add support for another two VVCs, thus doubling the number of supported virtual channels, the required storage space is 10400 bits, a mere 1.56 % increase in area space.

It is safe to assume that the implementation of the wiring between the added components will only amount to less than a 3% additional increase from the expected storage cost, but we needed an actual HDL implementation to be sure. To do so we modified a

generic 5-stage wormhole NoC router architecture to add the Mask-Based functionality. In the succeeding figures we show the additional modifications required to add support for the VC Renamer Mask-Based mechanism . For brevity only a single router port is shown. Figure 4.2 shows a port with two physical virtual channels with 2 mapped VVCs on each one. For each port a VVC-to-PVC mapping table is required in order to direct the flits in the right PVC based on the VC Id of the arriving flit. For each PVC we require a subtractor which will perform the head/tail mask comparisons when a flit is departing and when credits are sent. We also require an entry in the credit round robin pointer list to allow credits to be sent in a round robin fashion as it was shown in the beginning of this chapter. For each supported VVC we require a k-bit VVC mask, where k is the number of buffer slots for the PVC buffer, to retain the the VVC occupied slots on each PVC.

In Figure 4.3 the red lines show the actions that take place when a flit arrives at the input multiplexer of the receiving port. When a flit arrives the VC ID feeds the VVC-to-PVC mapping table so that the flit can be stored in the correct PVC . The VC ID also feeds the multiplexer which selects the correct VVC Mask whose bit is set to 1. When the flit is stored the tail pointer value is incremented. In the said figure a flit arrives for VVC0 which is stored in PVC0.

In Figure 4.4 the red lines show the actions that take place when a flit departs from the port to travel to the downstream router. The figure shows the departure for a flit from VVC0. The value of the head pointer from PVC0 is used along with the value of the VVC0 Mask in the subtractor so that the comparison can take place. If the head mask value is greater than that of the VVC0 Mask (after the mask bit where the head pointer is reset) the flit is allowed to depart.

In Figure 4.5 the red lines show the actions that take place when the round robin mechanism for the credits is in use. In each cycle for every PVC the current VVC id from the Credit Round Robin Pointer List is used to select the appropriate VVC Mask which is then fed into the subtractor along with the tail pointer of that PVC. If the tail pointer is greater than the value of the PVC mask and it does not point to an occupied position, a credit ON signal is sent. In the shown figure, the credit check is performed on VVC1 for PVC0 while for PVC1 the credit check is performed on VVC2. For VVC0 and VVC2 a credit OFF signal is sent. When the credits are sent each entry in the Credit Round Robin Pointer List is incremented so that credits will be sent for the next mapped VVC during the next cycle.

After implementing the Mask-Based mechanism of VC Renamer in Verilog and synthesizing it using Synopsy's Design Compiler results showed that when 2 VVCs are mapped on a single PVC (thus doubling the number of supported VVCs when compared to the generic PVC) the MB implementation incurs minimal area and power overhead of 2.09% and 0.23%, respectively. When 3 VVCs are supported the cost is still quite low with only a 2.87% area overhead and 2.33% power overhead. While still quite low, we can see that to support 4 VVCs the area and power overheads begin to rise (5.36% and 3.71% respectively) which shows a non-linear increase as the number of supported VVCs increases. Of course there wouldn't be any need to map more

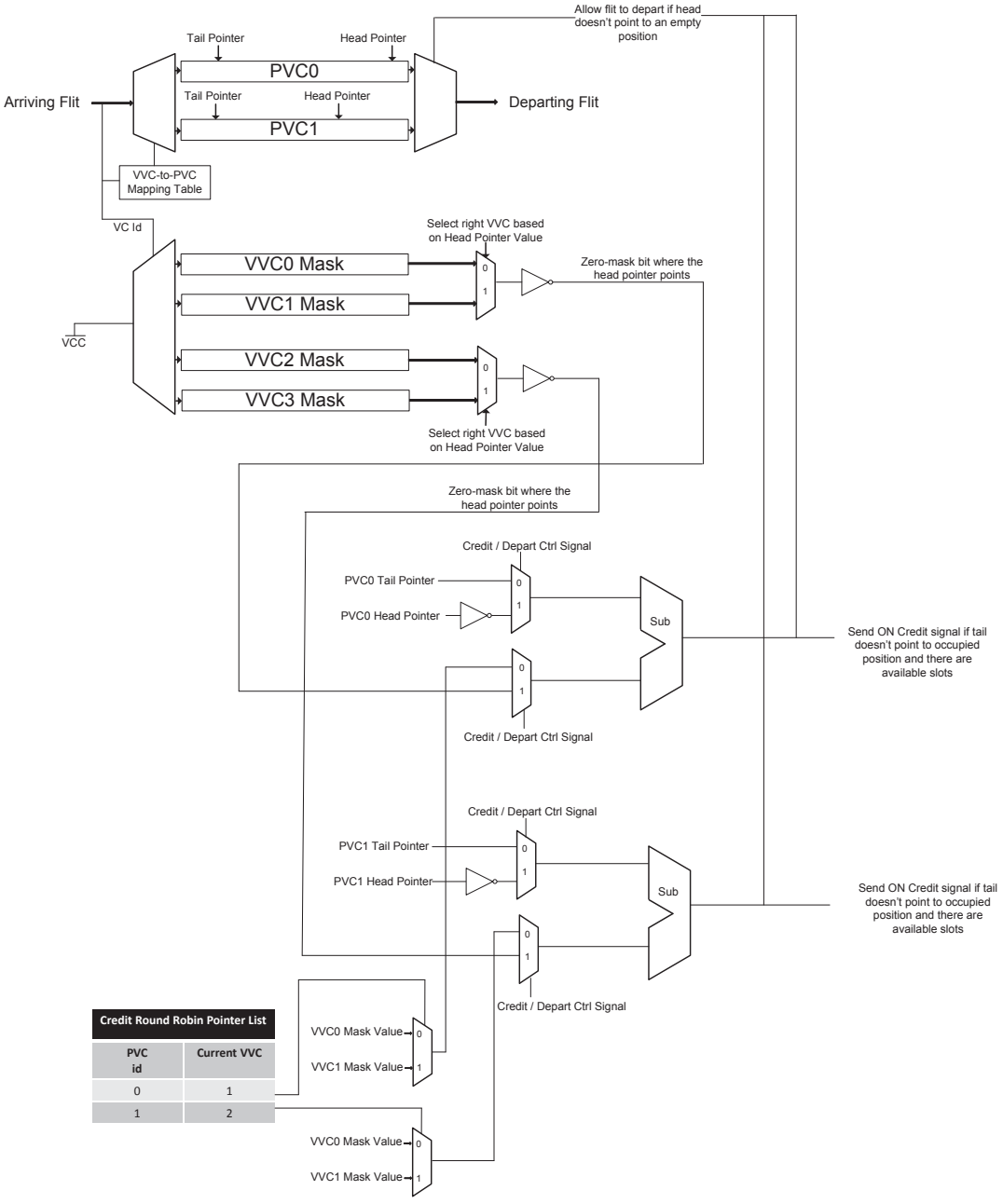


Figure 4.2: Mask-Based Hardware Architecture - A port with 2 PVCs and 2 mapped VVCs on each is shown.

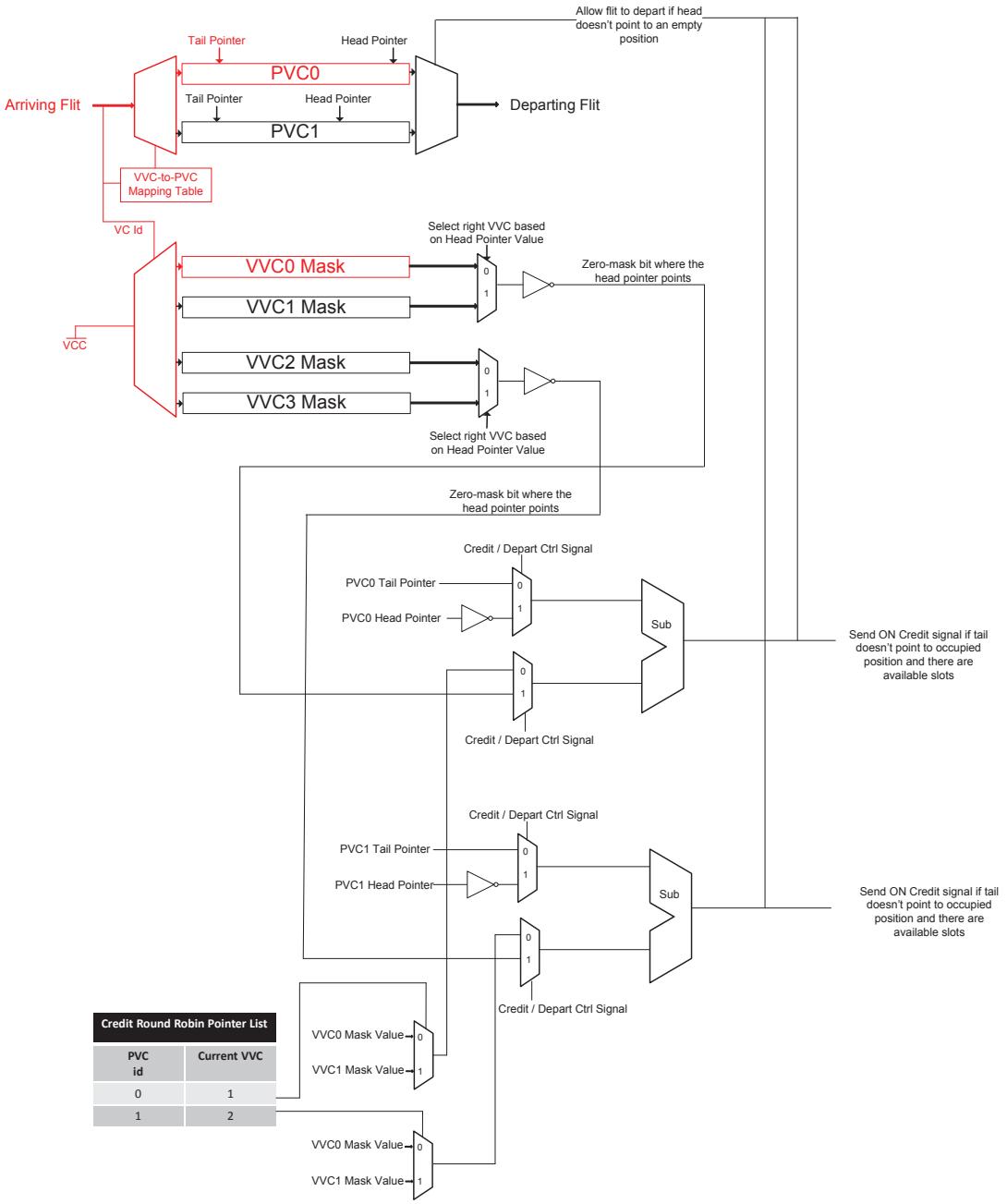


Figure 4.3: Mask-Based Hardware Architecture - Flit Arrival - Shows the actions performed when a flit arrives at the port. A port with 2 PVCs and 2 mapped VVCs on each is shown.

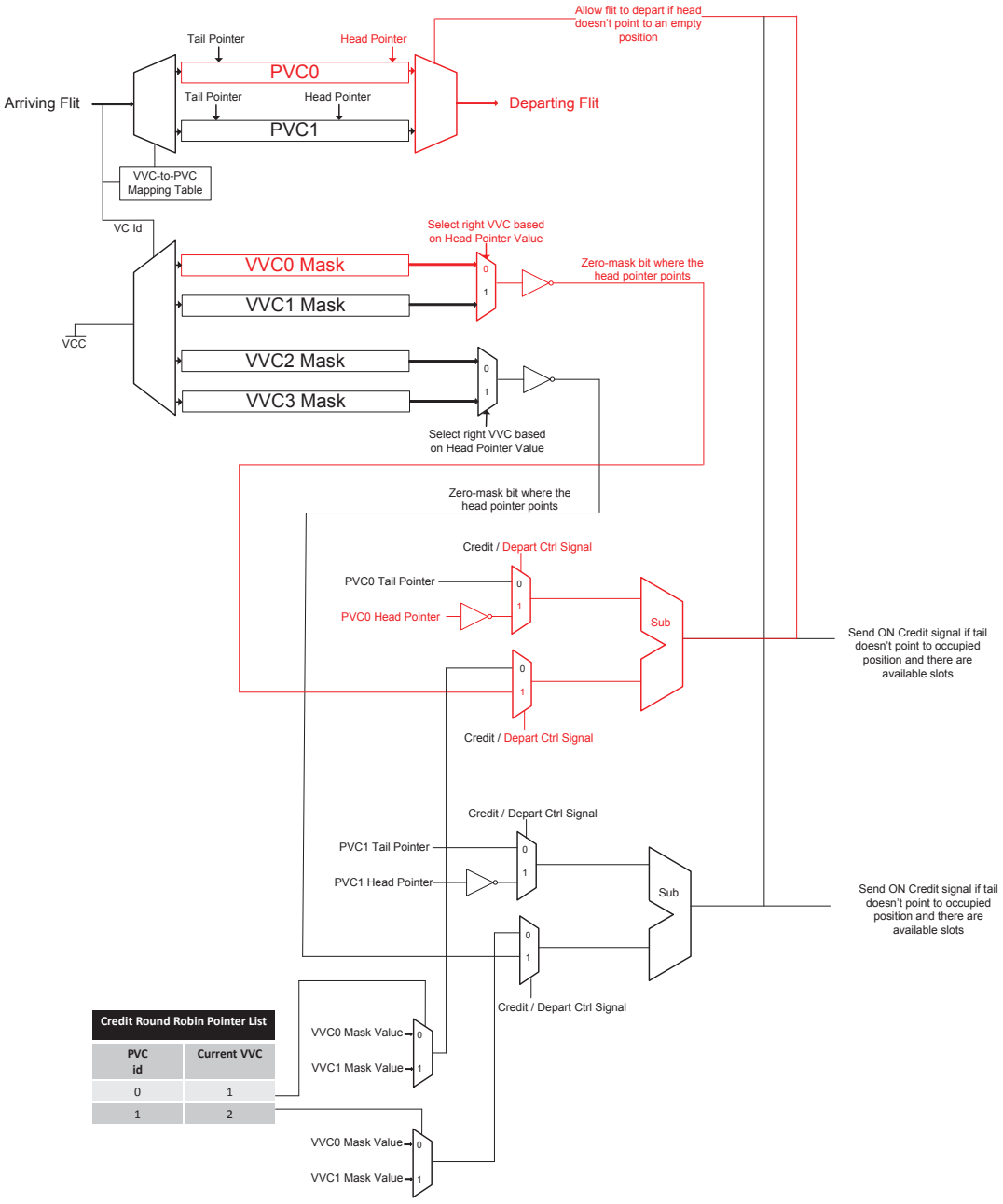


Figure 4.4: Mask-Based Hardware Architecture - Flit Departure - Shows the actions performed when a flit departs from the port. A port with 2 PVCs and 2 mapped VVCs on each is shown.

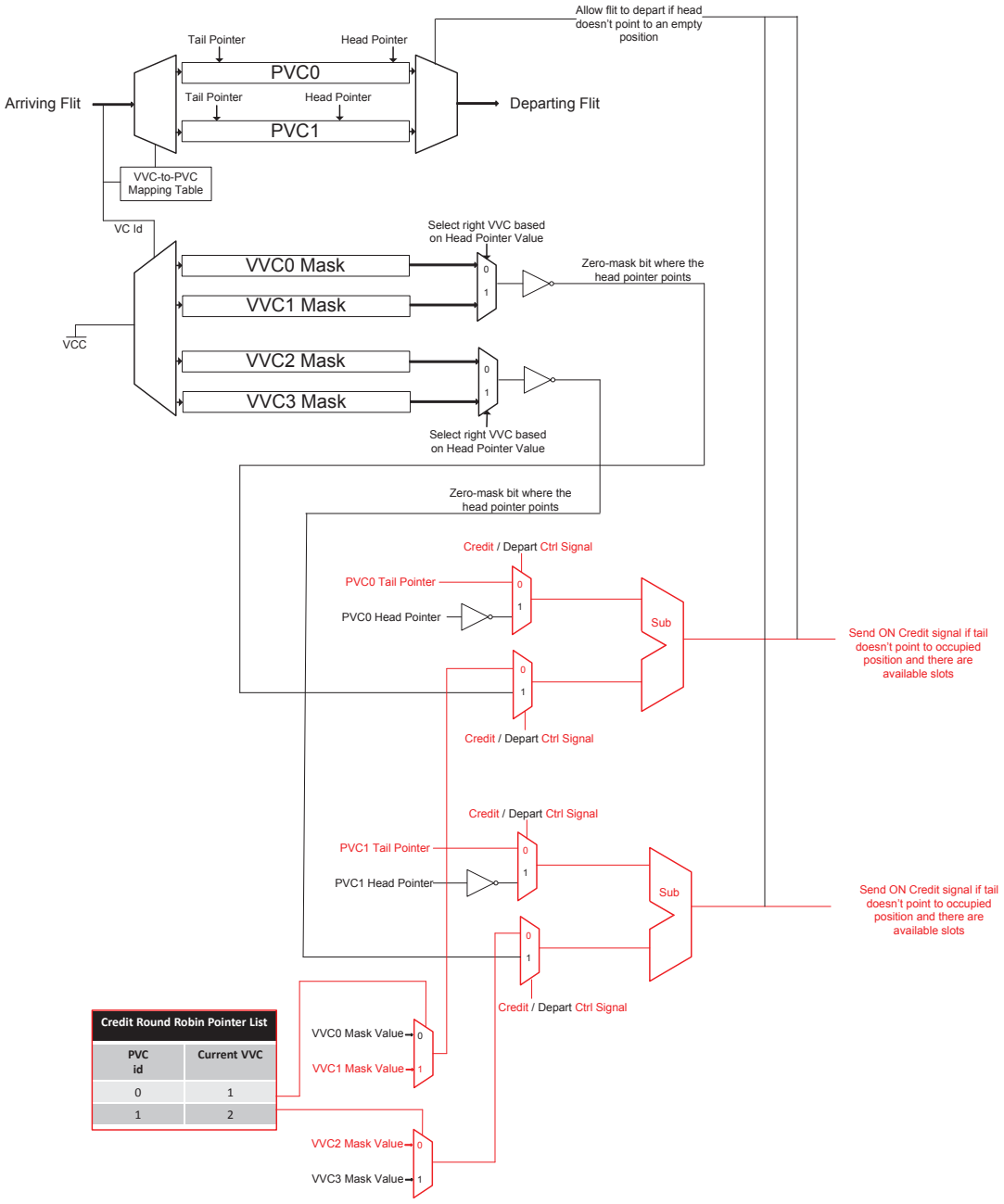


Figure 4.5: Mask-Based Hardware Architecture - Credit Mechanism - Shows the actions performed when credits are sent.

than 3 VVCs in a single PVC because that would hurt performance extremely. The important thing to notice is that the synthesis results showed that the critical path of the router was not affected by the Mask-Based implementation of VC Renamer. The critical path still lies within the VC Allocation (VA) stage, which is untouched by VC Renamer and all the new logic operates within the slack of the crossbar traversal and link-traversal/buffer-write stages. (Table B and C of figure 4.1)

5

VC Renamer - Linked-List-Based Implementation

5.1 Implementation in High-Level Simulator

Just as the Mask-Based mechanism the Linked-List-Based mechanism was also implemented in the cycle-accurate on-chip network simulator, Popnet A. This section presents, through a series of flow-control diagrams we are featured in C, the basic modifications needed in the high-level simulator in order to support the Linked-List-Based implementation of VC Renamer and to be able to turn-it on and off according to the needs of the network. Alongside the 5-pipeline stages modifications also had to be made to the way flits are handled when they arrive and to the way credits are sent.

- **Incoming Flits:** When a flit arrives at an input port, a check is made to see if VC Renamer is enabled on the selected port. If it is, an empty slot is acquired from the Free-Slot-FIFO List of the appropriate PVC using the VVC-to-PVC mapping table and the flit is stored. If the selected VVC, whose flit belongs to, had no prior flits stored within the buffer the VVC entry in the Front-of-VC List is updated using the flit position acquired from the Free-Slot-FIFO List. Otherwise the Back-of-VC List is used to index the PVC Pointer List and extend the linked-list. Then the Back-of-VC List is updated using the flit position acquired from the Free-Slot-FIFO List. If the VC Renamer bit is disabled the flit is simply stored in the regular PVC. (Figure C.1)
- **Router Pipeline Stage 1: Routing Decision:** During this stage a basic for loop begins examining all the router ports to see if routing needs to commence. For each of the ports if VC Renamer is disabled all the normal VCs are examined and if any of them are in the routing state (due to the arrival of a header flit), the appropriate routing function is called which returns all the port-VC pairs of the downstream router and stores them in a port-vc pair vector. If VC Renamer is enabled all the mapped VVCs are examined one by one. If any of them are in the routing state, the appropriate routing function is called which returns all the

port-VC pairs of the downstream router and stores them in a port-vc pair vector (Figure C.2)

- **Router Pipeline Stage 2: VC Arbitration VA1 Stage:** During VC Arbitration in the VA 1 Stage a loop examines all the router ports. If VC Renamer is disabled on the port, all the VCs are examined and if a particular VC is in the VC Arbitration state the vc selection function is called which uses all the port-vc pairs from the routing decision stage to find and return an available port-VC pair which is then stored in a table and used in the VA2 phase. If VC Renamer is enabled on the selected port all the VVCs are examined and if a particular VVC is in the VC Arbitration state the vc selection function is called which uses all the port-vc pairs from the routing decision stage to find and return an available port-VC pair which is then stored in a table to be used in the VA2 phase. When all the ports are examined, the simulator moves onto the stage 2 of the VC Arbitration phase. (Figure C.3)
- **Router Pipeline Stage 2: VC Arbitration VA2 Stage:** During the second stage of VC Arbitration all of the ports are being examined one by one. For each port all of the VCs (where VC Renamer is disabled) and all of the VVCs (where VC Renamer is enabled) are checked to see if they managed to acquire an output VC during the VA1 Stage. PopNET uses a random selection algorithm to grant a VC/VVC an output port. The algorithm basically examines if the output VC chosen by a VC/VVC is available and if it is, it's assigned to that VC/VVC and its state is updated to SW AB to be able to move to the Switch Arbitration pipeline stage. (Figure C.4)
- **Router Pipeline Stage 3: SW Arbitration SA1 Stage:** During the SW Arbitration, SA 1 Stage a loop examines all the router ports. If VC Renamer is enabled on the port all the VVCs are examined and if a particular VVC is in the SW Arbitration state, the assigned output VC is examined to see if there are available credits and if there are the VC number is stored inside a table which is used when SA2 commences. If VC Renamer is disabled on the port all the VCs are examined and if a particular VC is in the SW Arbitration state, the assigned output VC is examined to see if there are available credits and if there are the VC number is stored inside a table which is used when SA2 commences. When all the ports are examined, the simulator moves onto the stage 2 of the SW Arbitration phase. (Figure C.5)
- **Router Pipeline Stage 3: SW Arbitration SA2 Stage:** During the SW Arbitration SA 2 Stage a loop examines all the router ports and for each output port only one request can be satisfied with a random VC/VVC being selected. The winning request updates the state of the VC/VVC from SW AB to SW TR to allow the flit to move to the outbuffer stage. (Figure C.6)
- **Router Pipeline Stage 4: Flit Outbuffer Stage:** In the flit outbuffer stage the VCs/VVCs of each port are examined to see if they are in the switch traversal

state. If that is the case the flit is removed from the VC and added to the appropriate output buffer. The ports are examined one by one and if VC Renamer is disabled on a port, all of its VCs are examined and if a particular VC is in the SW Traversal state, a flit is removed from the VC and added to the appropriate output buffer. If the removed flit is at its destination it is consumed by the router and if its the tail flit the VC is released. If VC Renamer is enabled on the selected port, all of its VVCs are examined and if a particular VVC is in the SW Traversal state, a flit is removed from the VC and added to the appropriate output buffer. When a flit is removed the flit position is pushed into the Free-Slot-FIFO List entry of the appropriate PVC. If the flit is a tail flit the Front-of-VC List and Back-of-VC List entries are invalidated. If the flit isn't a tail flit the head pointer value is used to index the PVC Pointer List so that the Front-of-VC List can be updated with the next departing flit for that VVC. When all the router ports are examined the simulator proceeds to the Flit Traversal stage. (Figure C.7)

- **Router Pipeline Stage 5: Flit Traversal Stage:** During the flit traversal stage each output buffer (North, South, East, West) is examined and if its not empty, a flit us removed and sent to the appropriate downstream router. When all the output buffers for all the ports are examined a router cycle ends. (Figure C.8)
- **Credit Mechanism:** It's important to remember that the generic NoC router implemented within the high-level simulator originally used counter-based credits. For VC Renamer to work an ON-OFF credit mechanism was required where in each cycle a router sends an ON signal for each of its VCs to its downstream router if it can accommodate at-most one flit and an OFF signal if it can't. An extra process is added to the pipeline of the router which is performed in parallel with the basic router functions where in every clock cycle ON-OFF credits are being sent based on the slot availability of each VC. For each port where VC Renamer is disabled, each VC is examined and if it can accommodate one flit it sends an ON credit signal, otherwise it sends an OFF credit signal. As it's shown in the figure that the check performed is not for just one available slot but for four, in order to accommodate the four cycle turn-around time needed for a flit to arrive at the downstream router. If VC Renamer is enabled on a port and there's only one mapped VVC on a particular PVC the same procedure is used. For PVCs which have more than one VVC mappings, credits are sent in a round-robin fashion for each of the mapped VVCs. i.e. for three mapped VVCs it would be: VVC0-ON VVC1-OFF VVC2-OFF, VVC0-OFF VVC1-ON VVC2-OFF, VVC0-OFF VVC1-OFF VVC2-ON... (Figure C.9)

5.2 Implementation in HDL Language

In order to quantify the actual hardware area and power overheads of the Linked-List-Based Implementation in comparison to the generic NoC architecture, as well as verify

Table A: Linked-List-Based Implementation Storage Cost

Generic: 2 8-slot 128-bit VCs	Generic	VC Renamer – Linked List		
		2 VVCs /PVC	3 VVCs /PVC	4 VVCs /PVC
	10240 bits	10900 bits **6.44%**	10990 bits **7.32%**	11080 bits **8.20%**

Table B: Linked-List-Based Implementation – Required Hardware Area Cost

Generic: 2 8-slot 128-bit VCs	VC Renamer – Linked List		
	2 VVCs /PVC	3 VVCs /PVC	4 VVCs /PVC
	7.71%	**8.89%**	**11.37%**

Table C: Linked-List-Based Implementation – Required Hardware Power Cost

Generic: 2 8-slot 128-bit VCs	VC Renamer – Linked List		
	2 VVCs /PVC	3 VVCs /PVC	4 VVCs /PVC
	1.51%	**3.04%**	**4.67%**

Figure 5.1: Linked-List-Based Storage Cost - Cost of the Linked-List-Based Implementation compared to a Generic NoC Router as the number of mapped VVCs increases

that the critical path of the router remains the same we had to get actual results about the area costs, power costs and timing results for our implementation. To get an initial feel of the cost of our implementation we computed the storage cost required for the Linked-List-Based approach compared to that of the Generic NoC Network cost.

Generic NoC Network Storage Cost

- Buffer Slots : # of ports X # of VCs X # of buffer slots X flit size

VC Renamer Linked-List-Based Storage Cost

- Buffer Slots : # of ports X # of VCs X # of buffer slots X flit size
- Free Slot FIFO and PVC ID List : # of ports X # of VCs X # of buffer slots X log(# of buffer slots) x 2
- Front/Back of VC List and VVC count : # of ports X # of VVCs X log(# of buffer slots) x 3

It can be seen from Table A of the figure 5.1 that a generic NoC network with 2 8-slot 128-bit PVCs requires 10240 bits of storage space. If we enable VC Renamer to add support for another two VVCs, thus doubling the number of supported virtual channels, the required storage space is 11900 bits, which amounts to a 6.44 % increase in storage space. This is greater than the storage space required for the Mask-Based Implementation but this is due to the fact that the Linked-List implementation targets not only fault-tolerance but also the upgradability of the NoC, where all of the network routers will be using the VC Renamer mechanism simultaneously.

It is safe to assume that the implementation of the wiring between the added components will only amount to less than a 3 % additional increase from the expected storage cost, but we needed an actual HDL implementation to verify our assumption. To do so we modified a generic 5-stage wormhole NoC router architecture to superimpose the Linked List-Based functionality. In the succeeding figures we show the additional modifications required to add support for the VC Renamer Linked-List-Based mechanism . For brevity only a single router port is shown. Figure 5.2 shows a port with two physical virtual channels with 2 mapped VVCs on each one. For each port we require a VVC-to-PVC mapping table in order to direct the flits in the right PVC based on the VC Id of the arriving flit. For each PVC we require a k-flit PVC IDs List which will keep the ordering of the flits. We also require a k-flit Free-Slot FIFO which will keep the empty positions of the PVC. For each supported VVC we require an entry in the Front-of-VC-List which shows the position of the first stored flit of that particular VVC which is to be used as the head-pointer when flits depart. For each VVC we also require an entry in the Back-of-VC-List which shows the position of the last stored flit of that particular VVC in order to update the PVC Ids list correctly when another flit arrives for a VVC.

In Figures 5.3 5.3 5.3 5.6 the red lines show the actions that take place when a flit arrives at the input multiplexer of the receiving port.

- Flit Arrival Figure 5.3 : When a flit arrives at the input multiplexer of the port, the first thing that needs to be done is to acquire the position where the flit will be stored. The VC Id is used in order to select the correct PVC Id which then feeds the Free Slot FIFO in order to acquire the value of the tail pointer.
- Flit Arrival Figure 5.4 : Afterwards the tail pointer is used along with the PVC Id in order to store the flit in the correct PVC.
- Flit Arrival Figure 5.5 : If the arriving flit is the first flit of that VVC no changes occur in the PVC IDs list. If it's not the PVC IDs list is updated by storing the tail pointer value on the position pointed by the Back-of-VC List of that particular VVC.
- Flit Arrival Figure 5.6 : If the arriving flit is the first flit of that VVC, the Front-of-VC List for that VVC is updated by storing the tail pointer position value. At the same time the Back-of-VC List is updated by storing the tail pointer position value of that VVC.

In Figures 5.7 5.8 5.9 the red lines show the actions that take place when a flit is selected to depart from the port.

- Flit Departure Figure 5.7 : When a flit is selected to depart from the port, the first thing that needs to be done is to acquire the position of the departing flit. The VVC Id is used in order to select the correct position from the Front-of-VC List which gives the value for the head pointer.

- Flit Departure Figure 5.8 : Afterwards the head pointer is used along with the PVC Id in order to allow the flit to depart from the correct PVC.
- Flit Departure Figure 5.9 : If the departing flit is the last flit of that VVC the Front-of-VC and Back-of-VC List entries are invalidated. No changes occur in the PVC IDs list. If it's not the last flit the Front-of-VC List entry is updated with the value acquired front the PVC IDs list by using the head pointer value as reference.

The credit mechanism implemented for the Linked-List-Based approach is similar to that of the Mask-Based approach. Credits are sent in a round-robin fashion for all the VVCs mapped onto a single PVC. So basically the only difference with the Mask-Based approach is that there is no need to perform any comparisons before sending an ON credit signal, so as long as there are slots available in each cycle at most one mapped VVC will send an ON credit signal.

After implementing the Linked-List-Based mechanism of VC Renamer in Verilog and synthesizing it using Synopsy's Design Compiler results showed that when 2 VVCs are mapped on a single PVC (thus doubling the number of supported VVCs when compared to the generic PVC) we have a minimal area and power overhead of 7.71% and 1.51%, respectively. These appear to be higher than those of the Mask-Based approach but within the +3% margin of the storage cost we speculated. When 3 VVCs are supported the cost increases slightly with an 8.86% area overhead and 3.04% power overhead. We can see that to support 4 VVCs the area and power overheads increase in a greater rate (11.37% and 4.67% respectively) which shows that the cost of the wiring for the supported VVCs increases in a non-linear pattern. The important thing to notice is that the synthesis results showed that the critical path of the router was not affected by the Linked-List-Based mechanism either. The critical path still lies within the VC Allocation (VA) stage and all the new logic operates within the slack of the crossbar traversal and link-traversal/buffer-write stages. (Table B and C of figure 5.1)

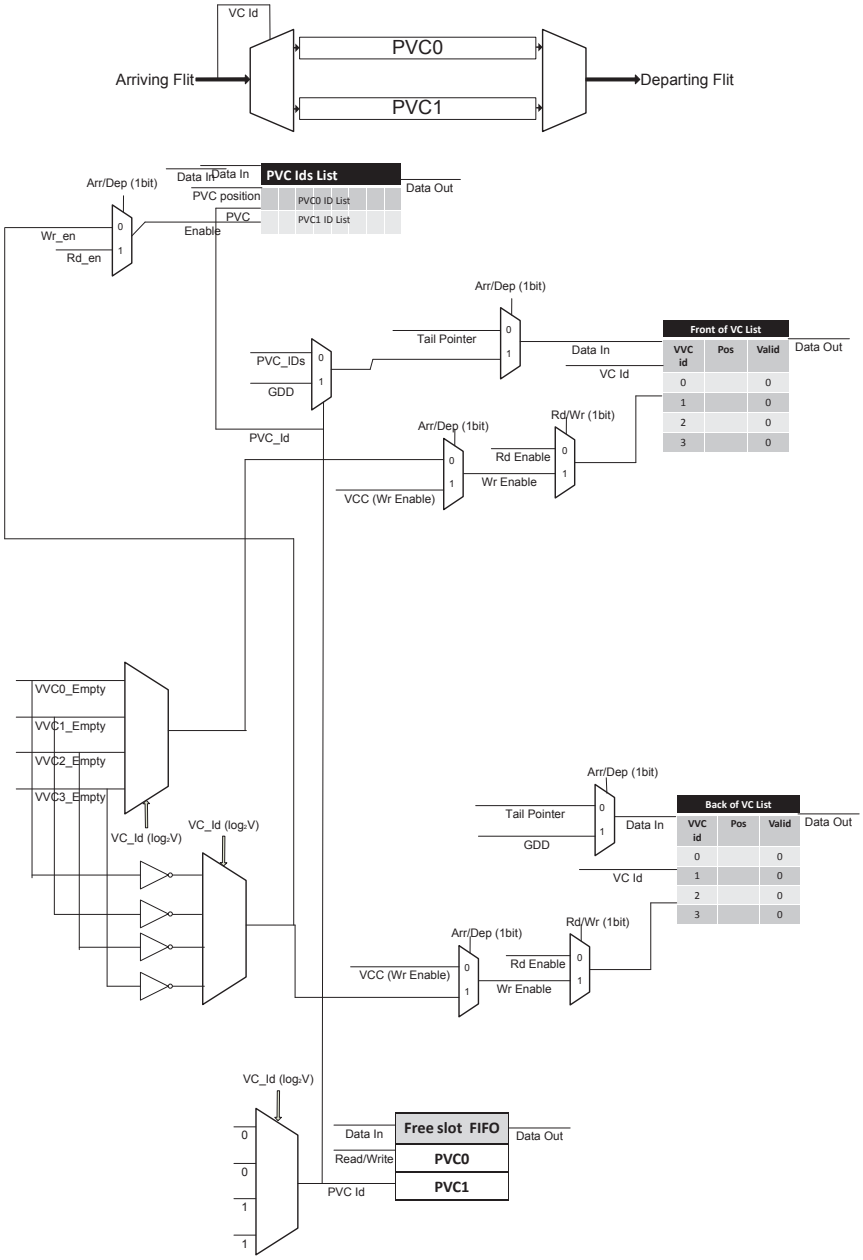


Figure 5.2: Linked-List-Based Hardware Architecture: figure shows the hardware needed in order to create a port with 2 PVCs and 2 mapped VVCs on each PVC.

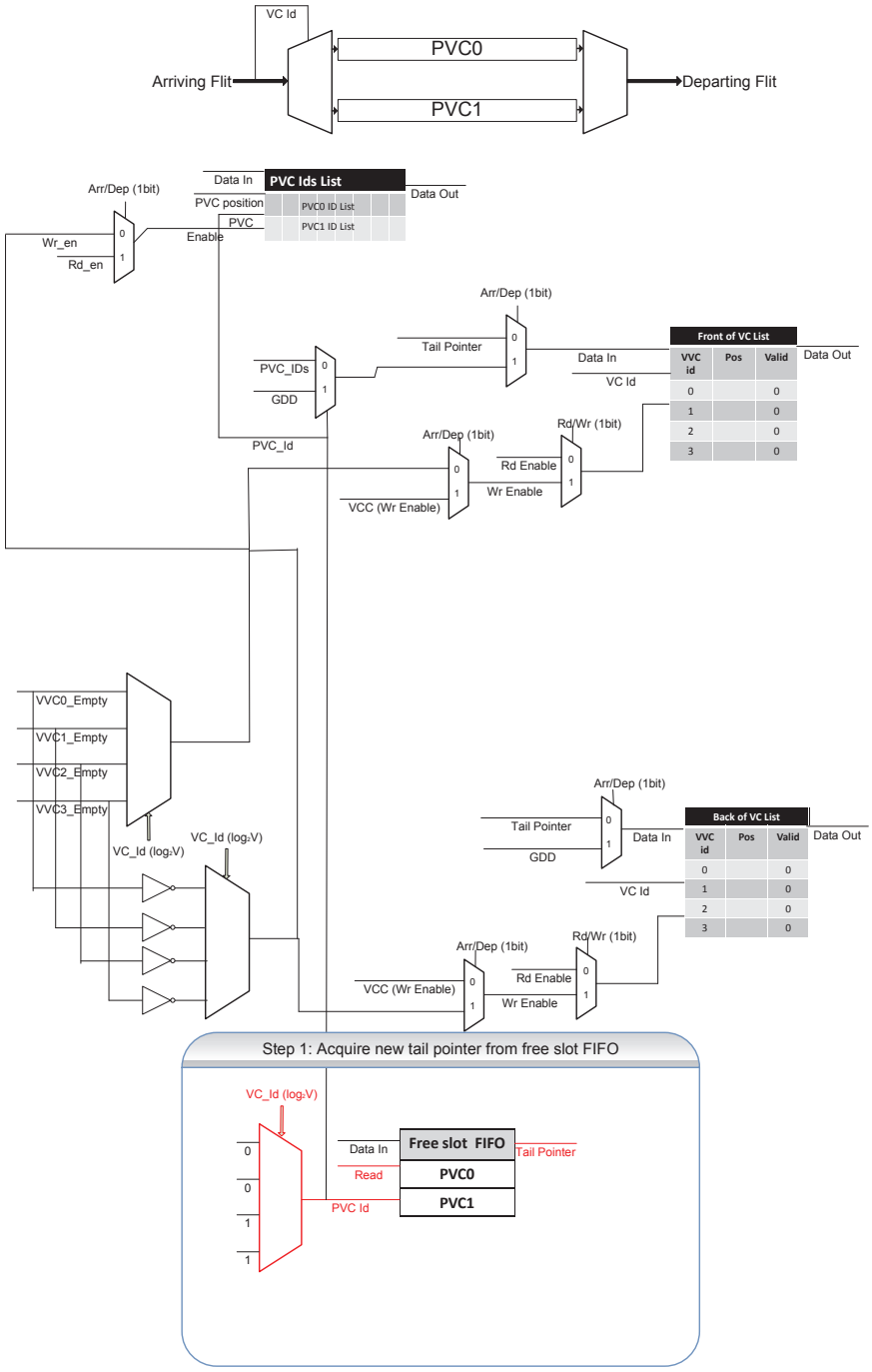


Figure 5.3: Linked-List-Based Hardware Architecture - Flit Arrival - Step 1: Acquire new tail pointer from Free-Slot-FIFO List

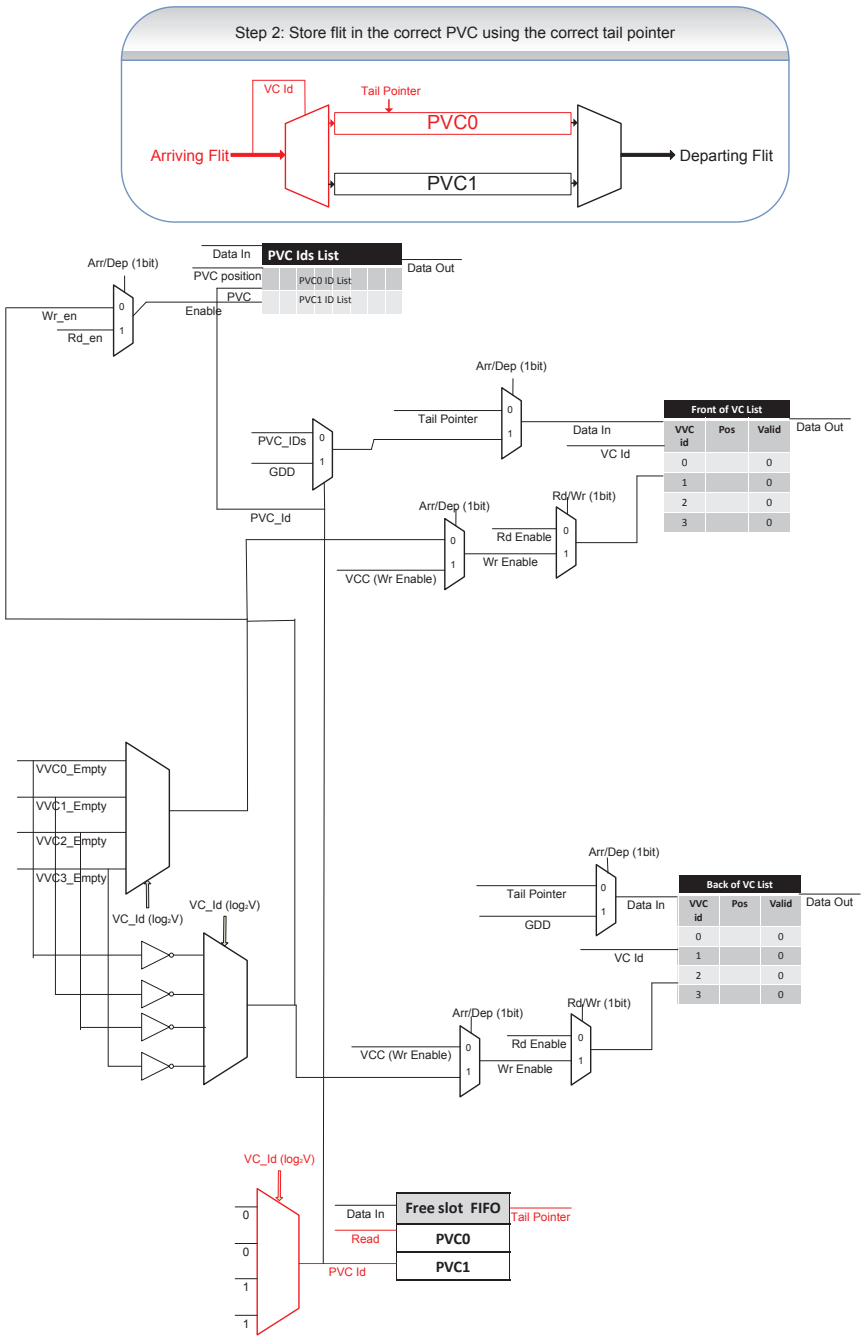


Figure 5.4: Linked-List-Based Hardware Architecture - Flit Arrival - Step 2: Store flit in the correct PVC

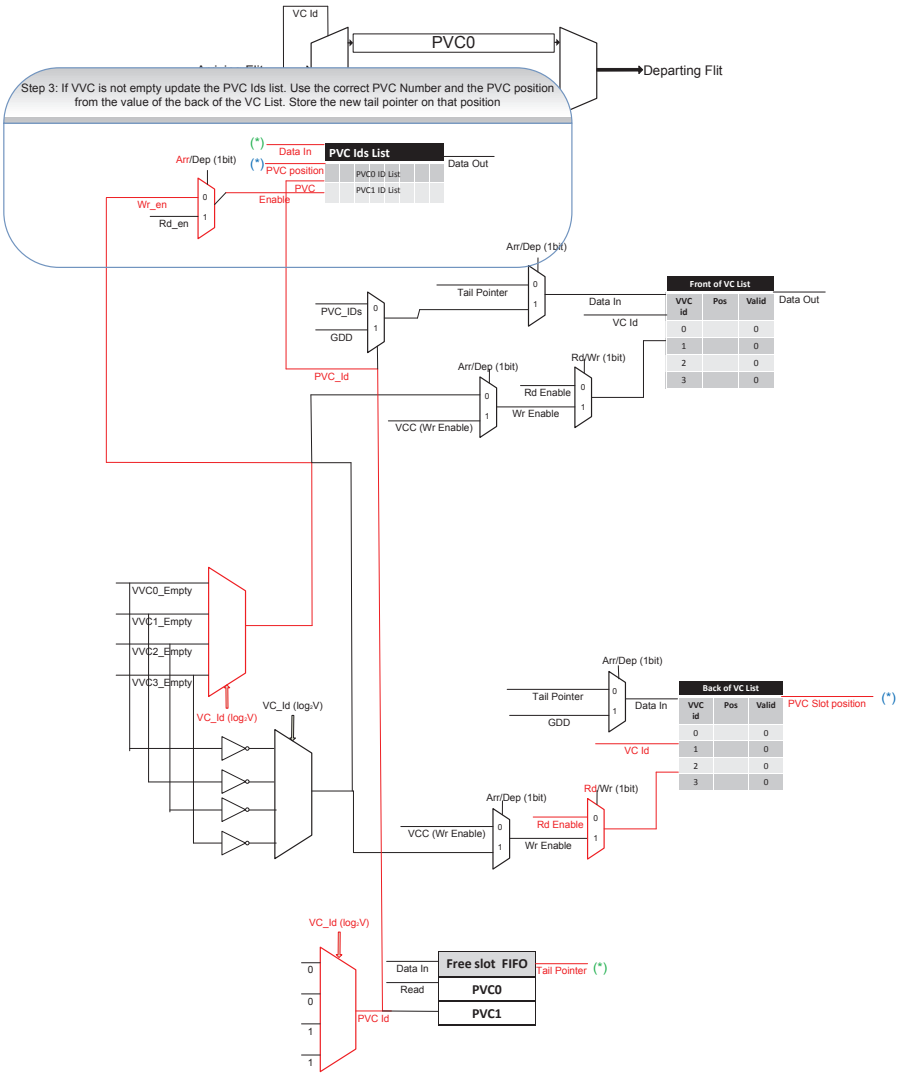


Figure 5.5: Linked-List-Based Hardware Architecture - Flit Arrival - Step 3: Update PVC IDs List using Back-of-VC-List Poistion

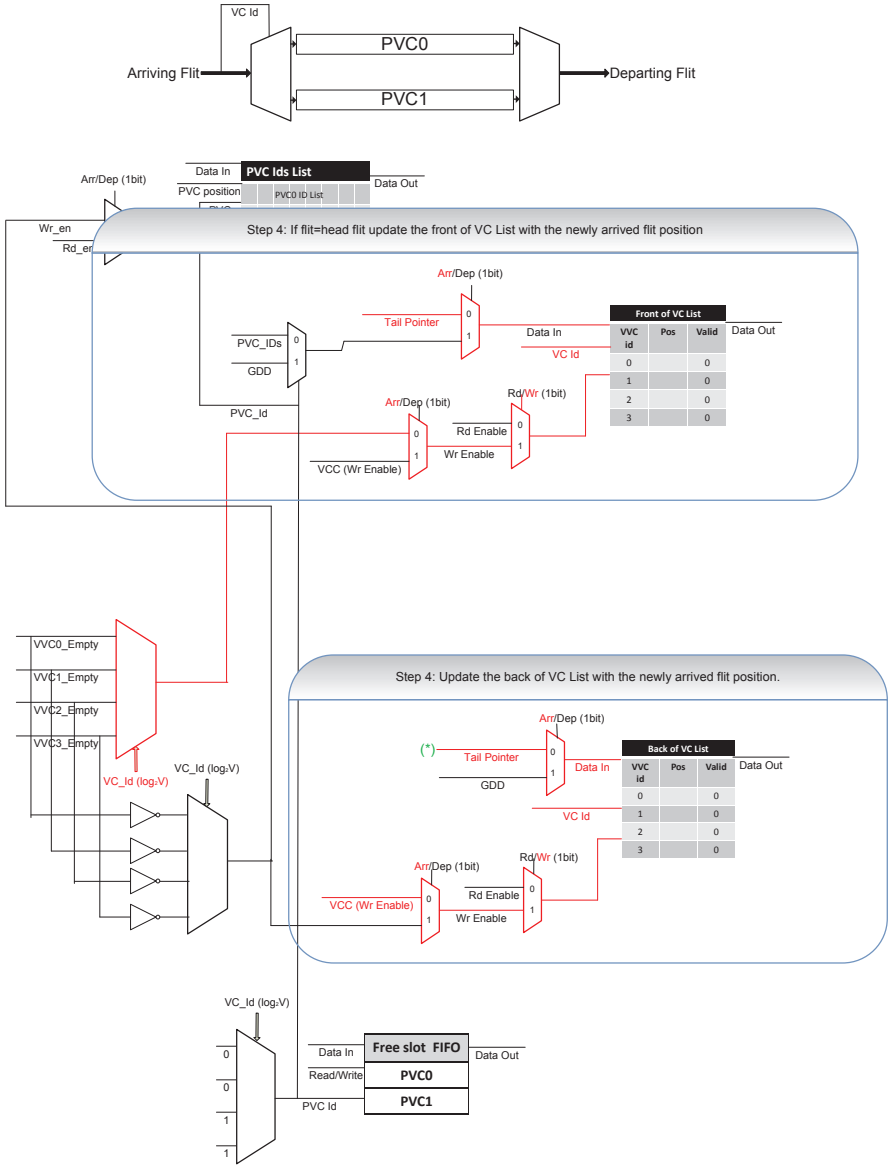


Figure 5.6: Linked-List-Based Hardware Architecture - Flit Arrival - Step 4: Update Back-of-VC List using Tail Pointer. If flit=head flit update Front-of-VC List using Tail Pointer

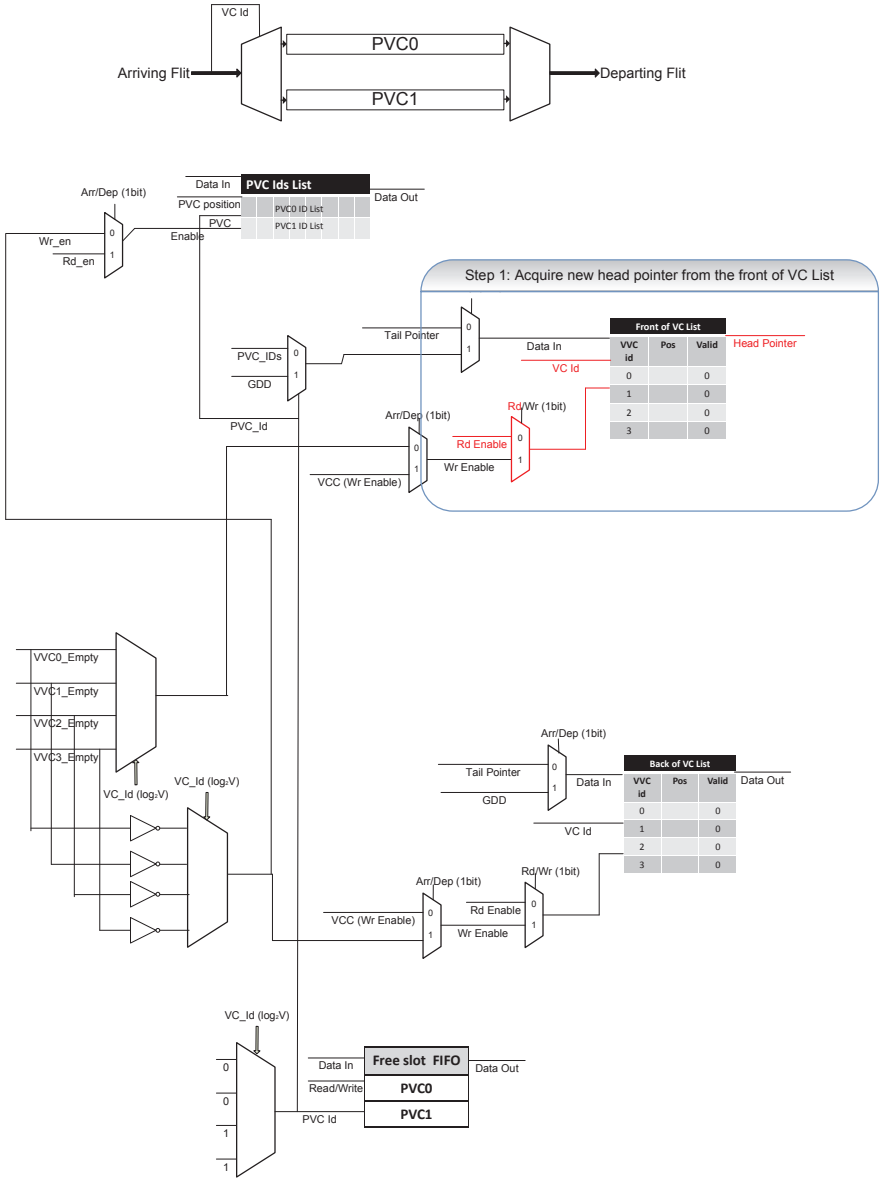


Figure 5.7: Linked-List-Based Hardware Architecture - Flit Departure - Step 1: Acquire new head pointer from Front-of-VC List

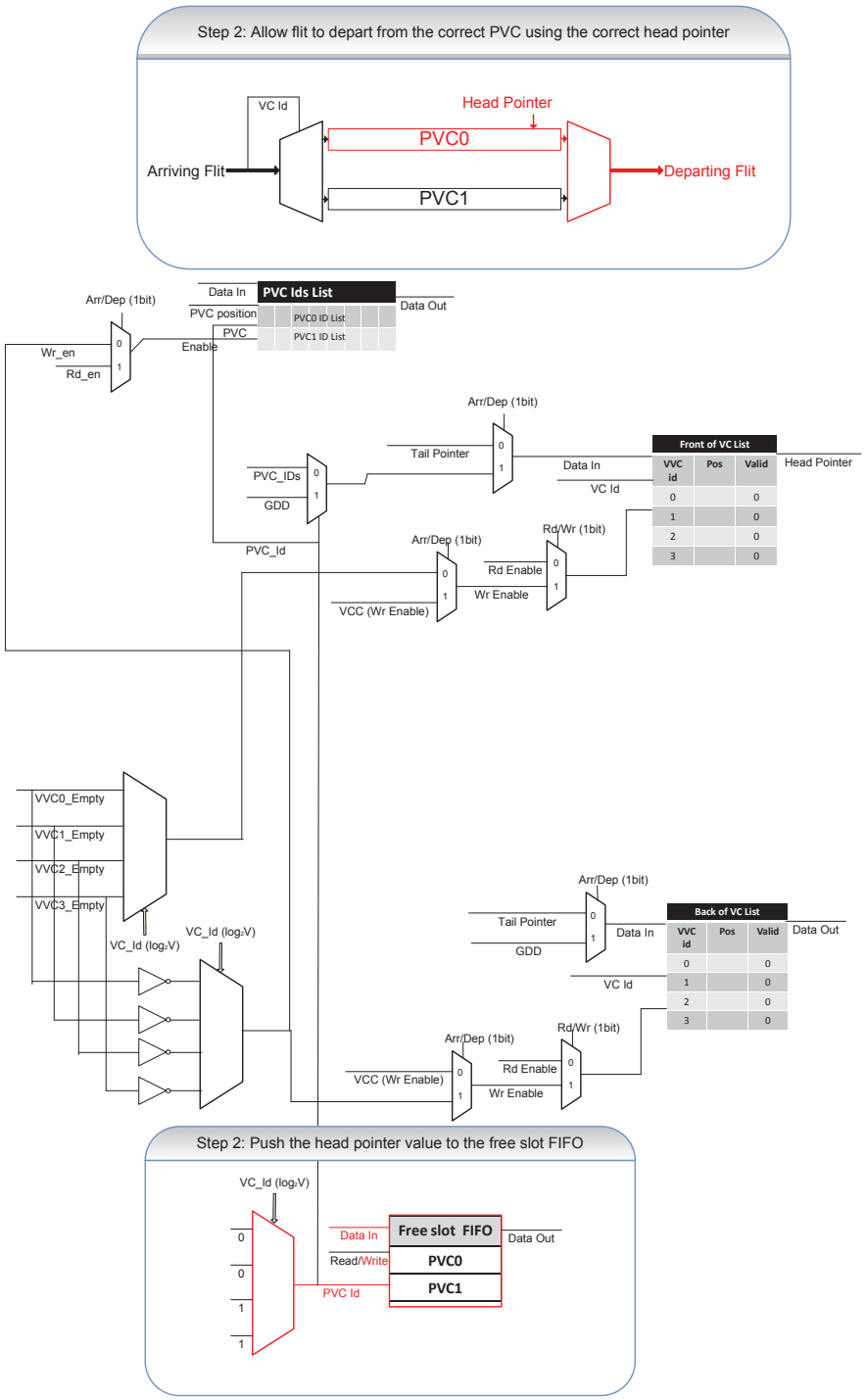


Figure 5.8: Linked-List-Based Hardware Architecture - Flit Departure - Step 2: Allow flit to depart from the correct PVC and push head pointer position to Free Slot Fiflo List

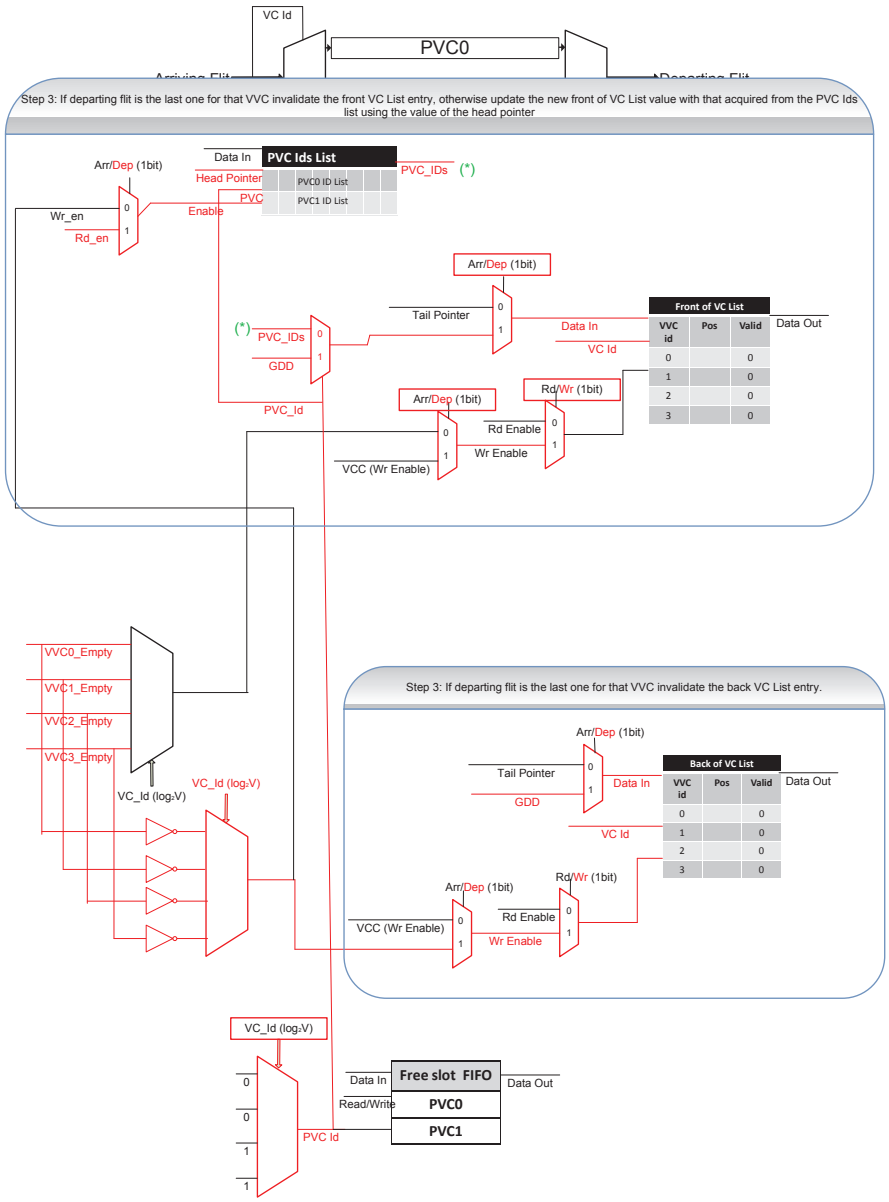


Figure 5.9: Linked-List-Based Hardware Architecture - Flit Departure - Step 3: If flit = tail flit invalidate Back and Front of VC List entries, else update Front-of-VC List entry by referencing the PVC IDs List

6

Simulations - Results Analysis

6.1 Simulation Platform

Both incarnations of VC Renamer were implemented within the cycle-accurate NoC simulator analyzed in section A which operates at the granularity of individual micro-architectural components. The simulations assume wormhole switching, 4-stage pipelined routers, and deterministic XY routing. Each router consists of five physical ports : North, South, East, West, and the local processing element. Every simulation runs for 1,000,000 clock cycles and each packet consists of five 32-bit flits. Our evaluation utilizes (a) **synthetic** Uniform Random (UR) traffic patterns in an 8×8 2D MESH network, and (b) traces from **real applications** running on the **TRIPS (40) NoC-based multicore processor**. The TRIPS processor includes a 4×10 mesh On-Chip Network (OCN) (40), which uses XY routing and *4 VCs per input port*. We use traces extracted from 11 representative benchmarks of the SPEC CPU2000 Suite (41) running on the TRIPS cycle-accurate simulator.

The *spatial distribution* of VC faults in the system is inspired by the model in (42). We define a *VC fault* as the inability to use a VC within a router input port, because of faults to components that affect the VC functionality. We explore two distributions of spatial VC fault placement: (1) Random (RM), where VC faults are *uniform-randomly distributed* throughout the NoC, and (2) Hotspot (HS), where VC faults are distributed only within a group of *spatially correlated* routers. In order to assess the robustness of VC Renamer, we vary the *percentage of faulty VCs in the whole NoC* from 1 to 10%. Each simulation was repeated 50 times and the results were averaged.

Finally, in order to evaluate the *hardware cost* of the proposed mechanisms, a conventional NoC router and both VC Renamer architectures were implemented in Verilog and **synthesized in Synopsys Design Compiler using 65 nm commercial standard-cell libraries**.

6.2 Results Analysis

6.2.1 Fault Tolerance Scenarios

We begin our evaluation with ***synthetic*** traffic patterns. We initially set the VC fault rate to 5% to see how VC Renamer fares as the traffic injection rate is varied. Each input port (in all designs) has four, 8-deep PVCs. We assume that the generic NoC design has *spare* VC buffers in *every* router input port to deal with VC faults. This, of course, amounts to an *enormous* overhead, which VC Renamer aims to eliminate by not relying on spare buffers *at all*. A fault in the MB and LLB designs is assumed to disable one of the 4 PVCs of an input port, thus *forcing two VVCs to be mapped to one of the remaining 3 PVCs*. The scenario in figures 6.1 and 6.2 assumes *RM* spatial fault distribution and compares the attained average network latency of VC Renamer to a generic NoC that is unaffected by the faults, because of the spare buffers (i.e., ideal scenario with immunity to faults). The MB and LLB implementations experience only 4.47% and 2.74% average drops in performance, respectively. *Throughput* decreases by only 4.96% and 0.52%, respectively. Similar trends are observed in the scenario of Figures 6.3 and 6.4, which assumes *HS* fault distribution. The MB approach exhibits worse performance, because some cycles are *skipped* (i.e., nothing happens) during operation, as a result of the Step 2 condition check of Algorithm 1 and Step 3 of Algorithm 2 (4). Here, the MB and LLB mechanisms experience 3.57% and 1.52% average declines in performance, respectively. The decrease in *throughput* is 5.22% and 0.75%, respectively.

The traffic injection rate is then set at 0.2 flits/node/cycle (in-between the zero-load and onset-of-saturation rates) and the VC fault rate is varied. Figures 6.5 and 6.6 illustrate the results assuming *RM* and *HS* spatial fault distributions, respectively. Note that the "Generic" latency in these figures is constant, since the generic design is unaffected by faults (ideal). Clearly, at low VC fault rates, the drop in performance is almost imperceptible with VC Renamer (as compared to an *ideal design unaffected by faults*). Even with 10% faulty VCs, the MB and LLB techniques experience modest 5.37% and 3.45% average drops in performance (over both spatial fault distributions), respectively.

Figure 6.7 summarizes results of trace-driven simulations of real applications running on the TRIPS processor. A VC fault rate of 5% and *RM* spatial fault distribution are assumed. On average, the MB and LLB implementations experience 6.11% and 1.80% decreases in performance, respectively, as compared to the ideal, fault-free setting. Clearly, both techniques are ***suitable for fault-tolerant designs***. If area/power overhead is an issue, the MB approach may be preferable, due to its lower overhead, as will be described shortly.

6.2.2 Upgradability Scenarios

In order to assess the *upgradeability* aptitude of VC Renamer, we run four different scenarios. In all the upgradeability scenarios VC Renamer is active in *all* router input

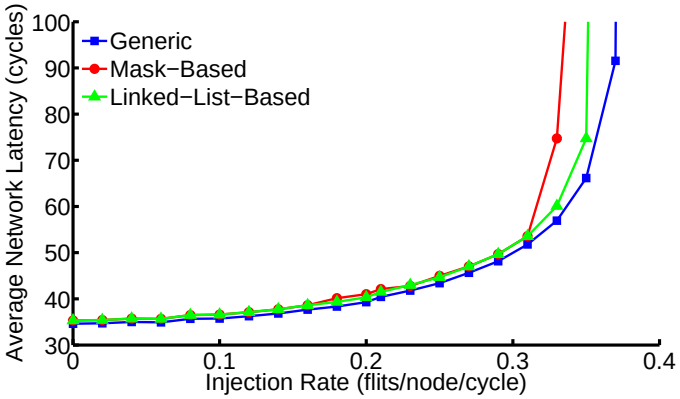


Figure 6.1: VC fault rate: 5%, *RM* spatial fault distribution, *UR* synthetic traffic

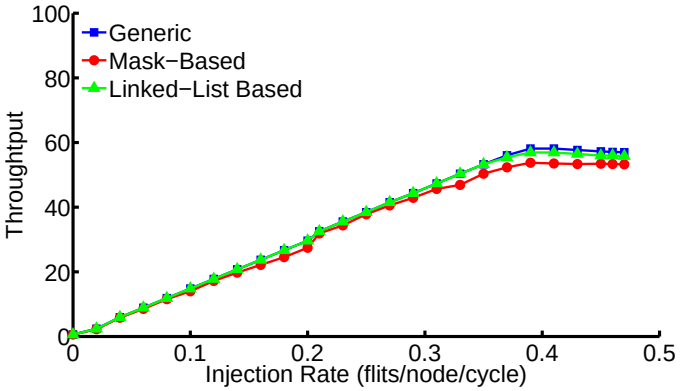


Figure 6.2: VC fault rate: 5%, *RM* spatial fault distribution, *UR* synthetic traffic

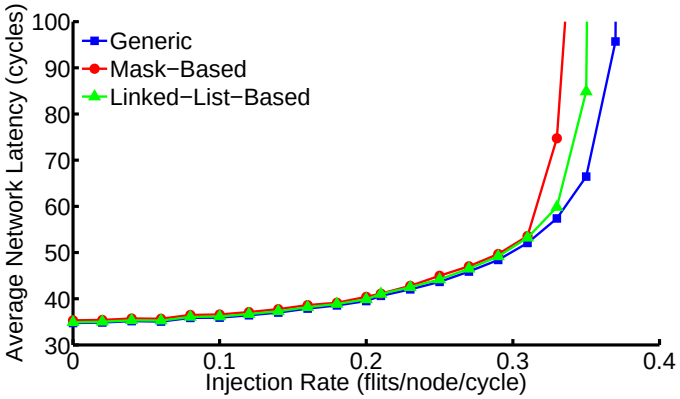


Figure 6.3: VC fault rate: 5%, *HS* spatial fault distribution, *UR* synthetic traffic

ports in the *entire NoC*. For fairness, all designs have the same total number of buffer slots (e.g., equal buffer space) per input port. In the first three scenarios we access the

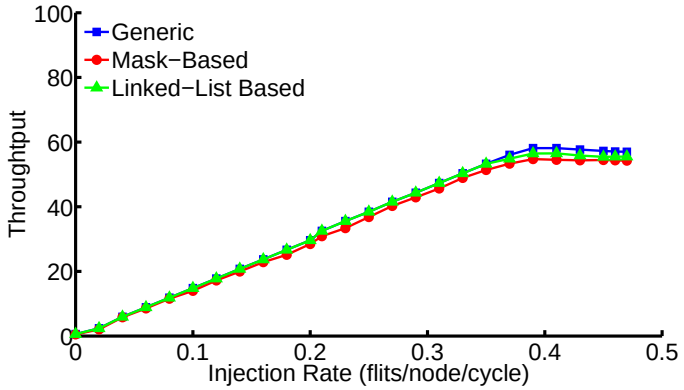


Figure 6.4: VC fault rate: 5%, *HS* spatial fault distribution, *UR* synthetic traffic

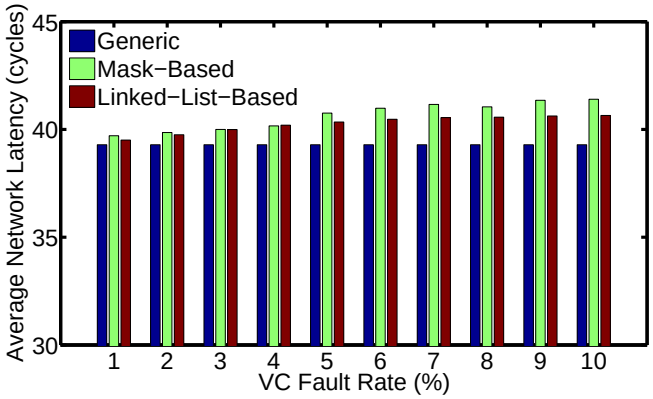


Figure 6.5: Injection Rate: 0.2 flits/node/cycle, *RM* spatial fault distribution, *UR* synthetic traffic

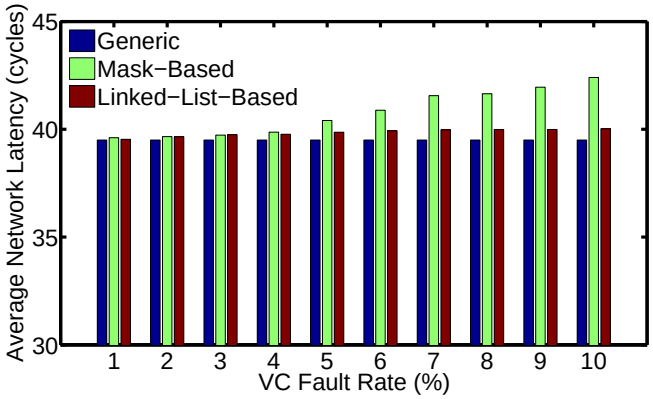


Figure 6.6: Injection Rate: 0.2 flits/node/cycle, *HS* spatial fault distribution, *UR* synthetic traffic

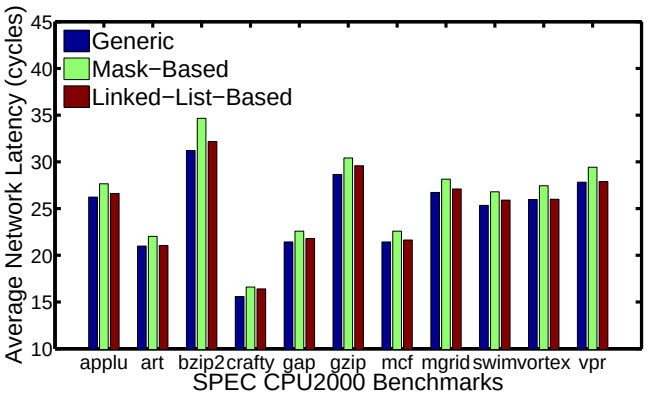


Figure 6.7: VC fault rate: 5%, *RM* spatial fault distribution, *Traces* from real applications

behavior of VC Renamer on an 8×8 2D MESH network using uniform ***synthetic*** traffic patterns. In the generic network of scenario 1 each router port has six 6-slot virtual channels. To keep the total number of buffer slots equal, the VC Renamer network routers have 4 9-slot PVCs in each port. This equals to two PVCs with two mapped VVCs and two PVCs with only one mapped VVC.

Figures 6.8 and 6.9 compare the latency and throughput of both VC Renamer implementations compared to that of the Generic NoC for the first upgradeability. The Mask-Based implementation suffers with a 13,39% increase in delay and a 2% loss in throughput, while the Linked-List-Based incurs a mere 3,22% in delay with a negligent drop of 0,0004% in throughput.

In the second upgradeability scenario each router port of the generic NoC has six 10-slot virtual channels while the VC Renamer NoC has 5 12-slot PVCs thus only one PVC has two mapped PVCs while the other four have only one mapped VVC. Figures 6.10 and 6.11 compare the latency and throughput of both VC Renamer implementations compared to that of the Generic NoC. We can see that in this scenario the Mask-Based implementation fares better with a a 4,69% increase in delay and a 2% loss in throughput. The Linked-List-Based implementation exhibits excellent performance with a delay of 0,13% and a 0,0001% loss in throughput.

The Generic NoC router in the third upgradeability scenario has 8 6-slot VCs while the VC Renamer routers have 6 8-slot PVCs, thus in this scenario we have two PVCs which have two mapped VVCs and four PVCs with only one VVC mapping. We can see from figure 6.12 that the MB-mechanism suffers from a 8,53% additional delay while the LLB-mechanism from a 3,49% delay. The graph in figure 6.13 shows that the MB-approach has a 3% drop in throughput and the LLB-approach a 0,0007% drop.

For the upgradeability scenario four we run the TRIPS (40) traces – which require 4 VCs/port – in a network with only 3 PVCs/port. While the MB implementation experiences a 15.52% average drop in performance (because of excessive cycle skips attributed to the condition checks of Algorithms 1 and 2), the LLB implementation

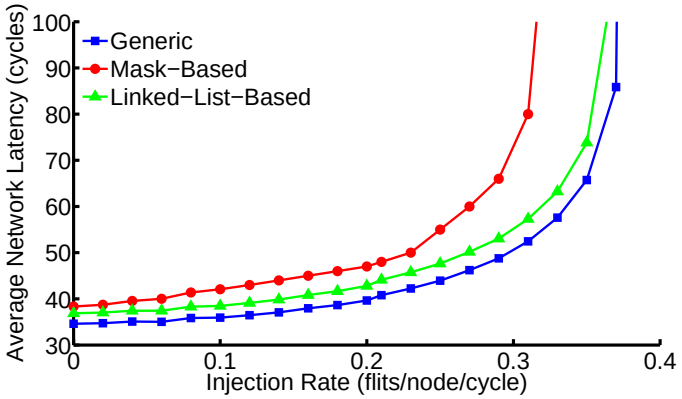


Figure 6.8: *Upgradeability Scenario 1 - Latency:* Generic:6 6-slot VCs - VC Renamer: 6 VVCs facilitated on 4 9-slot PVCs

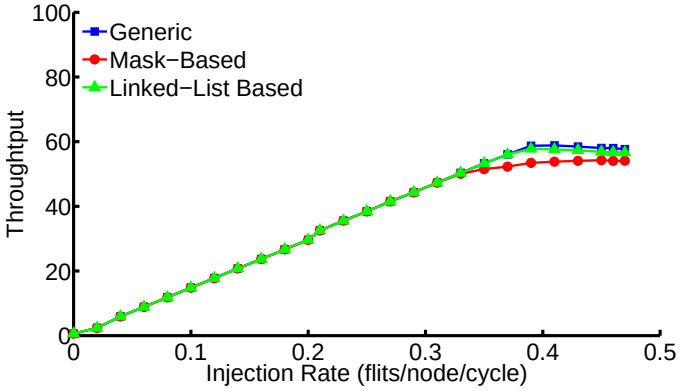


Figure 6.9: *Upgradeability Scenario 1 - Throughput :* Generic:6 6-slot VCs - VC Renamer: 6 VVCs facilitated on 4 9-slot PVCs

only suffers a 1.95% average decline.

It is interesting to note that deep PVC-buffers (with greater than 8-buffer slots) can better accommodate two VVCs, even in the case of the MB-mechanism. So both mechanisms can support upgradeability scenarios if they require one additional VVC. We can see from the scenarios above that when two PVCs accommodate two VVCs each, especially in the MB-mechanism, performance is clearly affected with more than 10% additional latency, while the LLB-mechanism has less than 5% additional latency. Also in an NoC network featuring more than four PVCs, the VC Renamer mechanism does not impact performance and in the LLB-mechanism its almost indistinguishable from the generic case. Clearly, the LLB technique is more ***suitable for upgradeability purposes***, due to its minimal impact on performance, even when used in all routers simultaneously.

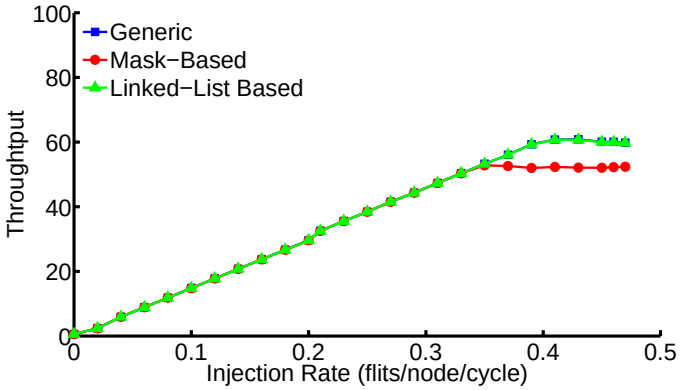


Figure 6.10: *Upgradeability Scenario 2 - Latency:* Generic:6 10-slot VCs - VC Renamer: 6 VVCs facilitated on 5 12-slot PVCs

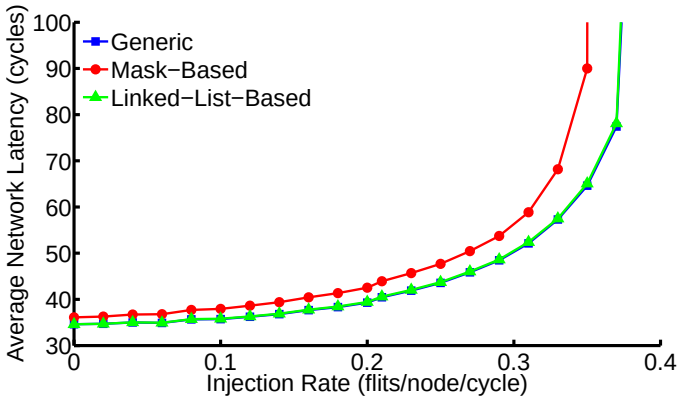


Figure 6.11: *Upgradeability Scenario 2 - Latency:* Generic:6 10-slot VCs - VC Renamer: 6 VVCs facilitated on 5 12-slot PVCs

6.2.3 Hardware Cost Comparison

In subsections 4.2 and 5.2 of chapters 4 and 5 respectiwely we saw the actual hardware implementations for the area and power costs for both VC Renamer designs individually. Table 6.1 summarizes the percentage area/power overhead of VC Renamer over a generic NoC design, as the number of mapped VVCs per PVC is varied for both designs. For example, if 2 VVCs are mapped per PVC, the number of supported VVCs is *double* that of the existing PVCs. We show results up to an extreme 4 VVCs/PVC to examine the *scalability* of the proposed design. Figures 6.15 and 6.16 show in graphs the percentage area/power overheads respectively. For *doubling* the number of supported VCs (i.e., a huge flexibility boost), the MB implementation incurs minimal area and power overhead of 2.09% and 0.23%, respectively, while the LLB technique incurs area and power overhead of 7.71% and 1.51%, respectively.

To actually quantify the area and power savings VC Renamer would provide we

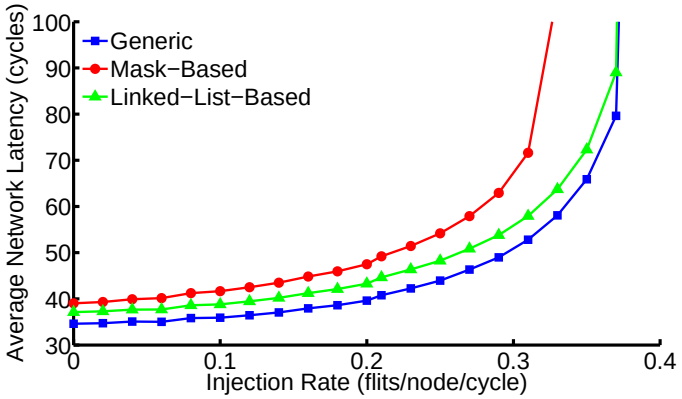


Figure 6.12: *Upgradeability Scenario 3 - Latency:* Generic:8 6-slot VCs - VC Renamer: 8 VVCs facilitated on 6 8-slot PVCs

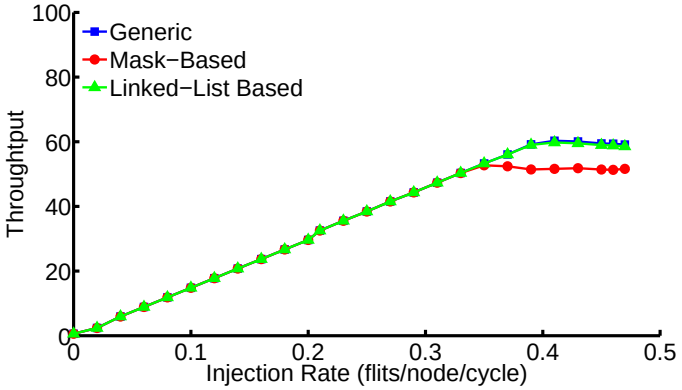


Figure 6.13: *Upgradeability Scenario 3 - Latency:* Generic:8 6-slot VCs - VC Renamer: 8 VVCs facilitated on 6 8-slot PVCs

compared a Generic NoC Router with 4,6,8 VCs with both implementations of VC Renamer with 4, 6, 8 VVCs. In order for the comparison to be fair, all implementations feature the same number of buffer slots. For example a PVC with 8 slots and 4 mapped VVCs is compared to a Generic NoC with 4 2-slot VCs. As it can be seen from the results of table 6.2 even with 4 mapped VVCs compared to a generic NoC architecture we have huge area and power savings. Compared to the Generic NoC architecture the Mask-Based implementation requires 75.69% less area and 8.75% less power while the Linked-List-Based implementation requires 66.53% less area and 6.40% less power. As the number of mapped VVCs increases the area and power savings are even greater with 411.63% and 296.52% area and power savings for the Mask-Based Implementation and 384.02% and 299.23% area and power savings for the Linked-List-Based implementation. When comparing the VC Renamer VVCs versus the Generic NoC VCs, both the VC Renamer implementation has enormous area and power savings.

More importantly, our synthesis results also showed that the **critical path of the**

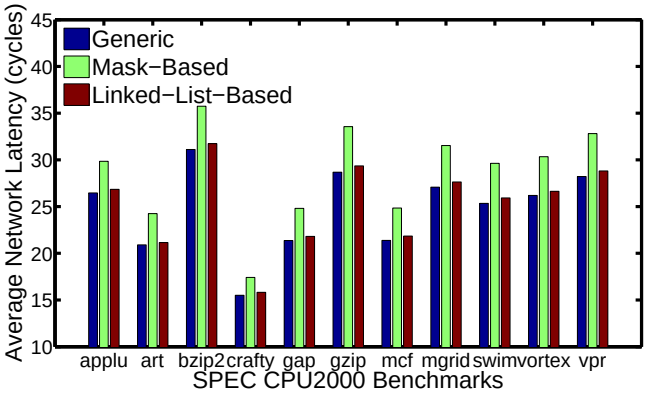


Figure 6.14: *Upgradeability Scenario 4 - Latency:*Generic:4 6-slot VCs - VC Renamer: 4 VVCs facilitated on 3 8-slot PVCs

Table 6.1: Hardware synthesis results: VC Renamer (2 PVCs) overhead over a generic NoC router (2 VCs) implementation

	% Area Overhead			% Power Overhead		
# of VVCs per PVC	2	3	4	2	3	4
Mask-Based	2.09	2.87	5.36	0.23	2.33	3.71
Linked-List-Based	7.71	8.89	11.37	1.51	3.04	4.67

router was *not affected* by either of the two VC Renamer architectures. The critical path still lies within the VC Allocation (VA) stage, which is *untouched* by VC Renamer (remember, VC Renamer simply changes the *mapping* of VVCs to PVCs; **it does not interfere with the operation of the arbiters**). All the new logic operates within the *slack* of the crossbar traversal and link-traversal/buffer-write stages. It is evident that overall the MB implementation costs are negligent while the LLB implementation costs are higher in order to ensure better performance for upgradeability scenarios.

6.2.4 Credit Mechanism

As mentioned at the end of Section 3.1, VC Renamer employs a round-robin credit dispatch mechanism for the VVCs mapped to a particular PVC. This mechanism sends credits to each of the VVCs on a cycle-by-cycle basis. Figure 6.17 analyzes the impact of this mechanism on average network latency, as compared to an *ideal* credit mechanism that dispatches credits to *all VVCs simultaneously* and only a VVC that could make use of the credits actually uses them. It can be seen that the lightweight round-

Table 6.2: Hardware synthesis results: VC Renamer Area and Power Savings over a generic NoC router implementation with an equal number of VCs and equal number of PVC Slots

	% Area Overhead			% Power Overhead		
# of VVCs	4	6	8	4	6	8
Mask-Based	75.69	237.82	411.63	8.75	104.20	296.52
Linked-List-Based	66.53	219.14	384.02	6.40	99.58	299.23

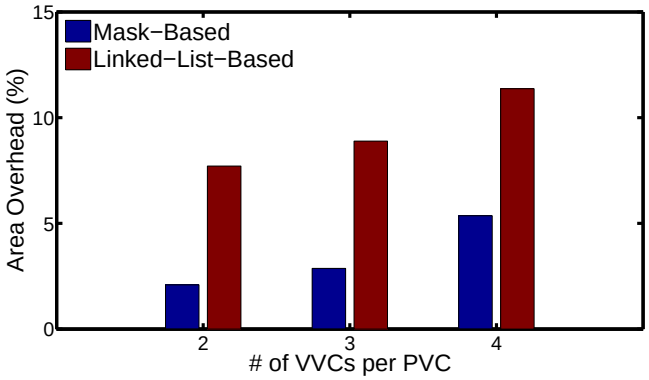


Figure 6.15: Hardware Synthesis: VC Renamer implementations area overhead compared to a generic NoC router

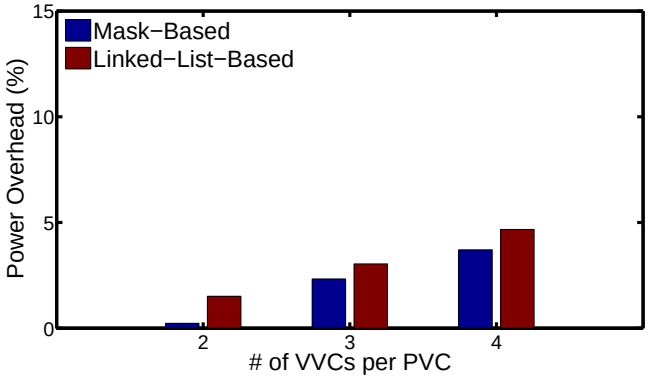


Figure 6.16: Hardware Synthesis: VC Renamer implementations power overhead compared to a generic NoC router

robin mechanism has negligible impact of 0.7% on performance when 2 VVCs are mapped to each PVC. For 3 VVCs/PVC, the latency increases by 2.7%, and for 4 VVCs/PVC it increases by 6.9%. Note, however, that at 2 VVCs/PVC, the number of supported VVCs already *doubles*. Hence, the credit mechanism has almost no impact on performance with this configuration.

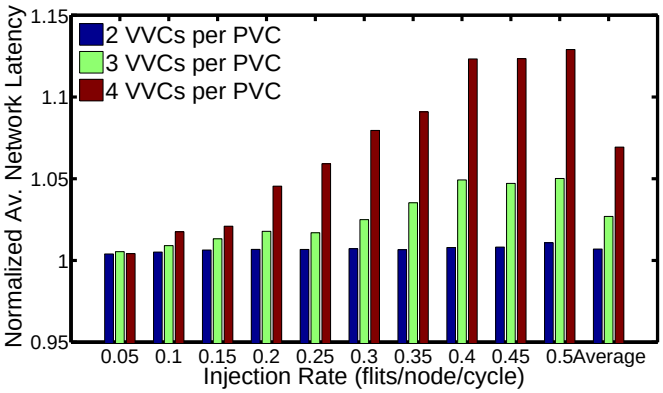


Figure 6.17: Comparison of average network latency achieved using the proposed round-robin (cycle-by-cycle) credit mechanism, as compared to an *ideal* mechanism that distributes credits *simultaneously* to all VVCs mapped to a particular PVC. The impact of the mechanism is assessed as the number of VVCs mapped to a PVC increases from 2 to 4. All results are *normalized to the setup with an ideal credit mechanism.*

7

Future Work - Conclusions

7.1 Future Work

It's evident from the results in Chapter 6, the premise of VC Renamer offers a valid, reliable and effective solution when: (1) A VC buffer/channel malfunction occurs and system functionality must be retained (2) A new routing algorithm, and/or new cache coherence protocol appears which requires a different number of VCs than the NoC network supports. We've shown that our mechanism works and our results are quite promising. We are now in the process of incorporating new notions which will extend its capabilities. Our near future plans feature three such possible directions: the incorporation of a mechanism to handle dynamic faults within the Network, to devise a mechanism in-order for one VVC to be able to be mapped on more than one PVCs and an exploration of the performance of VC Renamer using a variety of routing algorithms.

7.1.1 Handling Dynamic Faults

In section 6.2.1 of chapter 6 the faults we incorporated within the network in the fault-tolerance simulation scenarios were statically allocated at run-time so there was no need to worry about corrupted packets being routed within the network or even having packets lost never reaching their destination. Our current research plans are to devise a mechanism where VC Renamer can handle the presence of dynamic VC faults. We define a *dynamic VC fault* as the inability to use a VC within a router input port, because of faults to components that affect the VC functionality. When a dynamic fault occurs, the faulty virtual channel becomes inoperable and all the flits which were stored are rendered inaccessible. In order to be able to salvage these missing packets we devised a mechanism where the flits stored within a downstream router are present in it's upstream neighbour as well. So we basically maintain a copy of the flits in the neighboring router. When a fault occurs we notify the neighbouring router to resend lost flits and enable VC Renamer.

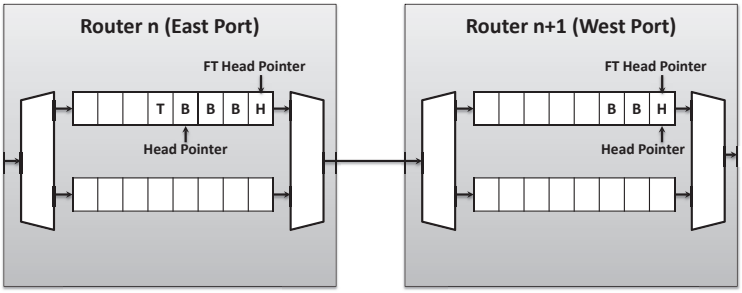


Figure 7.1: Generic NoC Architecture augmented to handle dynamic faults

Architecture

To accomplish this we require an extra head-pointer for every virtual channel, the fault-tolerant head pointer (FT head pointer) as shown in Figure 7.1. When router n (upstream router) sends a flit, it advances a second head-pointer (the actual head pointer), while another head-pointer (the fault-tolerance head pointer) remains in the same position. The flit arrives and is stored at router n+1 (downstream router). When router n+1 sends the flit towards router n+2, an OK message is sent from router n+1, back to router n in order to advance the fault-tolerance head pointer. If an error occurs within a virtual channel the faulty port sends a NOT OK message to its neighbouring port and enables the VC Renamer mechanism. Upon receiving that message the neighbouring port simply copies the value of the fault tolerance head pointer and overwrites the normal head pointer value. In order to prevent flits from getting overwritten before we are certain that they have departed from the downstream router, credit ON-OFF signals are sent based on the empty-free-slot availability using the fault-tolerance head pointer.

In order for the mechanism to work both the Linked-List-Based and Mask-Based implementations of VC Renamer need to be slightly modified to be able to handle dynamic faults. In order to handle dynamic faults the Mask-Based implementation requires an extra VVC Mask per VVC, the fault tolerant masks (FT Masks) as shown in figure 7.2. The basic algorithms we presented in 4 will also have to be slightly modified. More specifically when a flit arrives we update have to both the normal and FT VVC masks. When a flit departs the normal procedure is used where the bit on the normal VVC Mask is reset. In this implementation we send out credits based on the FT VVC Masks. When an OK message is received all that needs to be done is reset the rightmost bit of the FT VVC Mask, since the mask is being built from right to left. When a NOT OK message is received the normal VVC Mask is overwritten with the FT VVC Mask. Since the Generic NoC Architecture requires an extra set of head pointers, in the Linked-List-Based implementation we require an extra set of fault-tolerant entries in the Front-of-VC List, to store the fault-tolerant head pointer value as shown in figure 7.3. The basic algorithms we presented in 5 will also have to be slightly modified. During flit arrival no change is required, the normal value of the

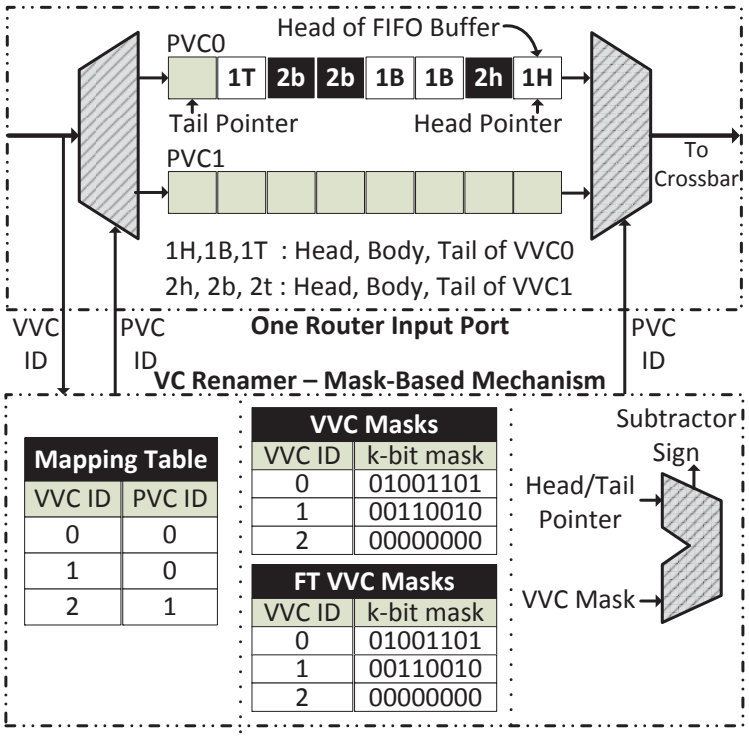


Figure 7.2: VC Renamer Mask-Based Architecture augmented to handle dynamic faults

Front-of-VC List is updated with a flit position from the Free-Slot FIFO List. Now when a flit departs we still use the Front-of-VC List entry value to index into the PVC Pointer and update Front-of-VC List Normal Entry but we do not push the previous position of the Front-of-VC List back to the Free-Slot-FIFO. When an OK message is received we use the Front-of-VC List FT value to index into the PVC Pointer and update Front-of-VC List FT entry and then push the previous FT value of the Front-of-VC List onto Free-Slot FIFO List. When a NOT OK message is received all that needs to be done is overwrite the Front-of-VC List Normal Value with the FT value.

Preliminary Results

To evaluate our mechanism we implemented three different architectures within our cycle-accurate NoC simulator. We implemented the mechanism to handle dynamic faults in a generic NoC router in order to evaluate it's performance in a fault-free network. Afterwards using the augmented network we implemented our two different architectures which were shown in figures 7.2 and 7.3 on-top of the mechanism to examine it's effects on latency and throughput in the presence of dynamic faults. All three mechanisms have been implemented but we are still in the beginning stages of our simulations. A preliminary scenario is presented here in order to evaluate is full-potential. Our scenario assumes wormhole switching, 4-stage pipelined routers, and

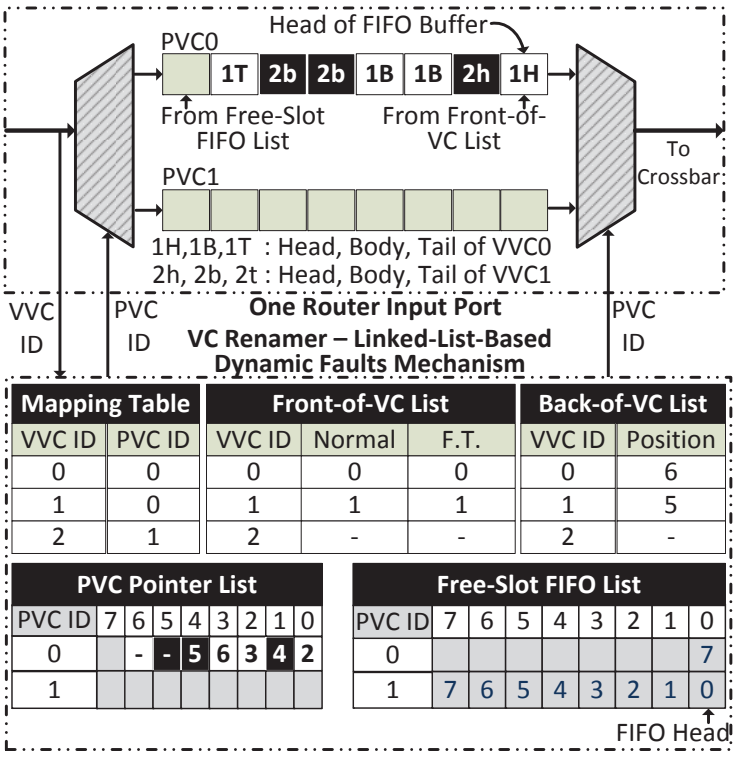


Figure 7.3: VC Renamer Linked-List-Based Architecture augmented to handle dynamic faults

deterministic XY routing. Each router consists of five physical ports and the generic router architecture features 4 10-slot virtual channels. In the VC Renamer implementations when a fault is materialized one of the four VVCs breaks down and is mapped onto another PVC. We run our simulation for 1,000,000 clock cycles using Synthetic Uniform Random traffic patterns in an 8×8 2D MESH network. The latency and throughput graphs appear in figures 7.4 and 7.5 respectively. From the said figures the blue line represents the generic NoC router architecture without any prior modifications. The red line represents the augmented generic NoC router architecture with no faults present in the network which uses the extra fault tolerant head pointer to make sure that in case an error occurs within a VC buffer no flits will be lost. The green line represents the Linked-List-Based mechanism augmented to handle dynamic faults in a network with 5% dynamic random faults. Finally the yellow line represents the Mask-Based mechanism augmented to handle dynamic faults in a network with 5% dynamic random faults. As it can be seen from the said figures the Generic router architecture saturates at 0.35 flits/node/per cycle. The FT mechanism without any faults at 0.30 flits/node/per cycle, the Linked-List-Based at 0.27 flits/node/per cycle and the Mask-Based at 0.20 flits/node/per cycle. Its evident that even in a fault-free environment our mechanism suffers with an additional 6.87% latency below saturation point and a

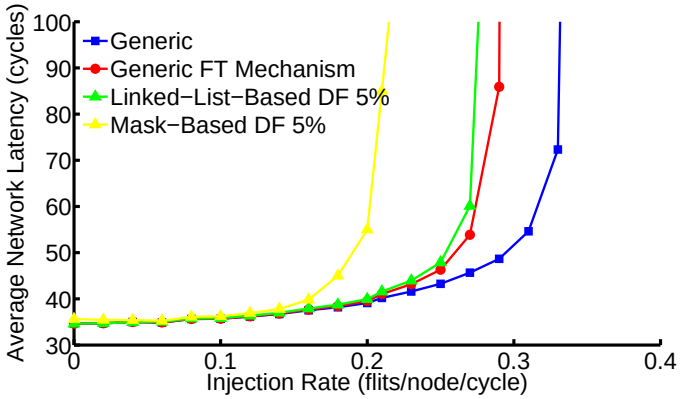


Figure 7.4: *Dynamic Faults Scenario - Latency:* Comparison of a Generic NoC, a Generic NoC augmented to handle dynamic faults and the VC Renamer mechanism under 5% dynamic faults

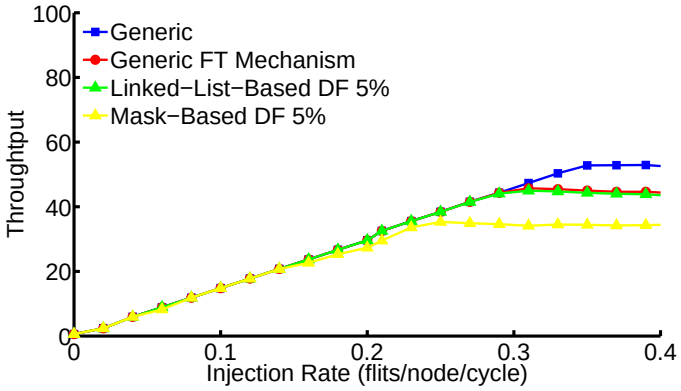


Figure 7.5: *Dynamic Faults Scenario - Throughput:* Comparison of a Generic NoC, a Generic NoC augmented to handle dynamic faults and the VC Renamer mechanism under 5% dynamic faults

5.38% drop in throughput overall. The Linked-List-Based implementation suffers with an additional 7.00% latency below saturation point and a 5.95% drop in throughput overall. Results are almost identical to the fault-free environment even at 5% faulty VC. It appears that the Mask-Based implementation is not resilient to handle dynamic faults and suffers the most with an additional 15.92% latency below saturation point and a 16.27% drop in throughput overall. It's evident that by sending the credits to the downstream routers using the FT VVC Masks extremely limits the performance of the network. We are still at the preliminary stages of our evaluation and there might be NoC configurations which will shed a different light in our innovative mechanism. The architecture presented might also be altered if we devise a better methodology to handle dynamic faults.

7.1.2 Mapping one VVC across multiple PVCs

As it was seen in Chapter 3, the VC Renamer mechanism implements a virtual VC which is directly mapped onto one physical VC. It is our belief that if we can map one VVC onto more than one PVCs in order to utilize buffer slots which aren't being used by the other VVCs will minimize the performance loss of the Mask-Based mechanism and even show the Linked-List-Based mechanism to be superior to the generic NoC. To be able to accomplish that, a control logic would need to be implemented which will keep track of incoming traffic and record the utilizations of the VVCs in each router port. These traffic measurements, which will be taken using well defined equations, will be used to decide if a VVC can 'borrow' slots from another PVC. Let's say we have a router with 2 8-slot physical virtual channels with one VVC mapped on each. VVC0 is used for data transfers and VVC1 for control messages. The control logic measuring traffic utilizations shows that the VVC1 is always half-empty, while VVC0 is almost always full. Since that is the case it could send a signal to map VVC0 on PVC1 and help the flow-control mechanism of the network.

7.1.3 Exploration of the performance of VC Renamer using various routing algorithms

Every simulation scenario we performed used the XY routing algorithm. This was to stress out the fact that when using deterministic routing algorithms a single fault within the NoC would render the network inoperable. The VC Renamer mechanism was able to keep the network from completely failing with a slight loss in performance even when deterministic routing algorithms were used. The XY routing algorithm is simple and effective but it lacks the adaptation which can be found in other routing algorithms, which can utilize traffic measurements and information from neighbouring routers. It is in our future plans to examine the performance of VC Renamer using a variety of routing algorithms both deterministic and adaptive.

7.2 Conclusions

Virtual channels are quintessential constructs in the correct operation of both the NoC *routing algorithm* and the CMP's *cache coherence protocol*. This thesis introduces the notion of **VC Renaming**, which enables the *further* ***virtualization of existing VC buffers***, in order to decouple the number of supported VCs in the system from the number of physically present VC buffers. The goals are (a) to enable the system to tolerate faulty VCs without reliance on expensive spare buffers, and (b) to accommodate routing algorithms and/or cache coherence protocols with varying VC requirements. Two different hardware implementations of the *VC Renamer architecture* are presented, which target different objectives (fault-tolerance vs. upgradeability). Both designs incur minimal hardware overhead and exhibit excellent performance without impacting the router's critical path. These results are very promising and demonstrate the viability of VC Renaming in future CMPs.

Appendix A

POP Net - A high-level cycle-accurate NoC Simulator

POP net is a cycle-accurate interconnection network simulator developed by Li-Shiuan Peh of Princeton University in 2000-2001. The simulator makes extensive use of the Standard Template Library (STL) which is a C++ library of container classes, algorithms, and iterators providing many basic algorithms and data structures useful in creating a network simulator. What makes it possible are the STL library containers which are highly parametrizable and can be used as templates for the various classes which will build up the NoC.

The basic router structure of a router in Popnet is comprised of two components the front end (input template) and the back-end (output template). The front end component contains the virtual channels, a table which records the current pipeline stage of each VC as well tables to store the result of the routing decision for each VC. The back end component contains the output buffers and counters which hold the state of the credits of the routers neighbors.

A.1 Front-end Router Component

As we have mentioned above the front-end component of Popnet contains the basic variables needed for the virtual channels. This is accomplished using STL vectors and pairs.

- **input [port] [vc] [vc size]** : a 3d vector container which contains the incoming flits
- **states [port] [vc]** : holds the current stage of the router pipeline that each VC is in. The stages which the VC can be in are:

  INIT : Initial Stage : Awaiting for flits to arrive

  ROUTING : Routing Stage : Routing must be performed on the header flit

VC AB : Virtual Channel Arbitration : VC arbitration must be performed on the header flit

SW AB : Switch Arbitration Stage : Switch arbitration must be performed on each flit.

SW TR : Switch Traversal Stage : Switch traversal must be performed on each flit.

- **routing [port] [vc] [direction-vc pairs]** : a 3d vector which after the routing is performed contains a series of pairs which hold the direction which the flits of a packet must follow as well as the number of virtual channels of the neighboring router for that direction. (direction-vc number)

- **crouting [port] [vc] [direction-vc pair]** : a 2d vector which after the vc arbitration is performed chooses one VC from the VC pairs of the routing vector, if any are available. This denotes the direction and the VC the flits of that particular packet will follow in the switch arbitration and switch traversal stages.

A.2 Back-end Router Component

The back-end component of the router functionality contains the output buffers and the variables which hold the credit state for each of the routers neighbors. Just as with the front-end component of the router this is accomplished using STL vectors.

- **counter [port][neigbouring vc]** : contains a counter with the available credits of the neighboring port for each of its VCs. Each counter is initialized to the buffer size of each VC. When a flit departs towards that direction for a particular VC the appropriate counter is decreased. Each time a flit from the downstream router departs it sends a message to the router informing it to increase the according counter by one.

- **usage [port][vc]** : contains the state of the neigbouring VCs. This can be either FREE or USED. If the VC is in the FREE state, the particular VC is a candidate for the VC arbitration stage. When a header flit arrives in the neigbouring VC, it sends a message and the usage vector changes into the USED state where it cannot participate to the VC arbitration stage. When a tail flit departs from the neigbouring VC, it sends a message and the usage vector changes into the FREE state

- **outbuffers[output port][outbuffer size]**: a 2d vector container within which the outgoing flits are stored before they traverse the link to reach the downstream router.

- **localcounter[output port]**: counter which holds the number of the available positions in the output buffers.

A.3 Router Pipeline

The router pipeline in Popnet contains 5 stages. These are the routing decision, virtual channel arbitration, switch arbitration, flit output buffer stage and flit traversal stages. In a router cycle these stages are performed for each router before advancing to the next cycle. The 2d vector states[port][vc] from the front-end component of the router dictates if any action is going to occur during that stage. For example if the virtual channel 2 of the north port of router 0,0 is in the VC AB stage then during the next routing cycle VC arbitration will commence for that particular VC. To ensure that when the VC arbitration stage ends and the state of that particular VC is updated, the switch arbitration doesnt execute for that particular VC, Popnet runs the router stages in reverse order.

- **Routing Decision Stage** : During the routing decision stage if a particular VC is in the ROUTING state, routing commences. The appropriate routing function is called, based on the routing algorithm specified from the command line options, and the routing function returns a set of pairs of all the VCs of the downstream router where the packet should follow (port, vc number). When routing completes the state of the VC is updated from ROUTING to VC AB and the routing pairs are stored inside the routing[port][vc][direction-vc pairs] vector.

- **Virtual Channel Arbitration Stage**: Virtual Channel arbitration succeeds the routing decision stage. This is broken down into two distinct stages VA1 and VA2. In VA1 all the input ports which are in the VC ARB stage request an output port. So each virtual channel for each port is examined to assert if they are in the VC AB state. If thats the case the vc selection function is called which uses the routing[port][vc][direction-vc pairs] vector to examine if any of the downstream routers VCs are available. This is done using the usage vector. All the VCs which manage to acquire a neighboring VC are stored within a map container which is one of the STD librarys features and is basically a sorted associative array of unique keys and associated data which in our case are the winning VCs which follow on to the VA2 stage. During the VA2 stage arbitration occurs where only one VC manages to acquire each output VC. The state of the VC winning the VA2 stage is updated from VC AB to SW AB and the selected routing port and output VC are stored within the crouting (chosen routing) vector.

- **Switch Arbitration Stage**: Switch arbitration is broken down into two stages, SA1 and SA2. During the first stage of switch arbitration all the ports and virtual channels of a router are examined and if a VC is in SW AB state the chosen routing vector is acquired from the previous stage via the crouting vector and if there are available credits both in the output buffer of the current router and credits from the downstream routers selected VC the request is stored within a map container which follows to the SA2 stage. During the SA2 stage only one output VC is selected and the winning request updates the state of the VC from SW AB to SW TR to be able to follow to the flit outbuffer stage.

- **Flit Outbuffer Stage**: During the flit outbuffer stage the VCs of each port are examined to see if they are in the switch traversal state. If that is the case the flit is removed from the VC and added to the appropriate output buffer. If the removed flit is at its destination it is consumed by the router and if its the tail flit the VC is released. Then a CREDIT message is sent to inform the neighboring router to increase its credit counter by one.
- **Flit Traversal Stage**: In the flit traversal stage each output buffer (North, South, East, West) is examined and if its not empty, a flit is removed and sent to the appropriate downstream router using a WIRE message.

A.4 Message Passing

Data and control signals in Popnet are transmitted using four kinds of messages: EVG, ROUTER, WIRE and CREDIT messages.

- **EVG messages** : messages used when a new packet is going to be injected into the network
- **ROUTER messages** : messages used to determine the pipeline stage for each router
- **WIRE messages** : messages used for routers to receive flits from other routers
- **CREDIT messages** : messages used for credits

A.5 Command Line Options

Popnet by itself is quite parametrable and thats why it stands out of other similar simulators. One can specify the various NoC parameters using the following command line.

Command file to run: ./isim -A 8 -c 2 -V 2 -B 6 -O 2 -P 5 -L 200 -f 4 -H 10 -F 32 -I /home/user/popnet/uniform/500/bench -p 3 -R 0 -S 0

- A 8 : determines the size of the network in each dimension
- c 2 : determines the dimensionality of the network
- B 12 : determines the input buffer size
- 0 2 : determines the output buffer size
- F 4 : determines the flit size
- L 1000 : determines the link length in run
- T 2000 : determines the simulation cycles

- -l /home/user/popnet/uniform/500/bench : specifies where the trace file is located
- R 0 : determines the routing algorithm used (R 0: is YX routing for 2 VCs per port)

The Popnet trace files follow the format: T sx sy dx dy n: where:

- T : packet injection time
- sx sy : the address of the source router
- dx dy : the address of the destination router
- n : packet size (number of flits)

A. POP NET - A HIGH-LEVEL CYCLE-ACCURATE NOC SIMULATOR

Appendix B

VC Renamer - MB Flow Diagrams

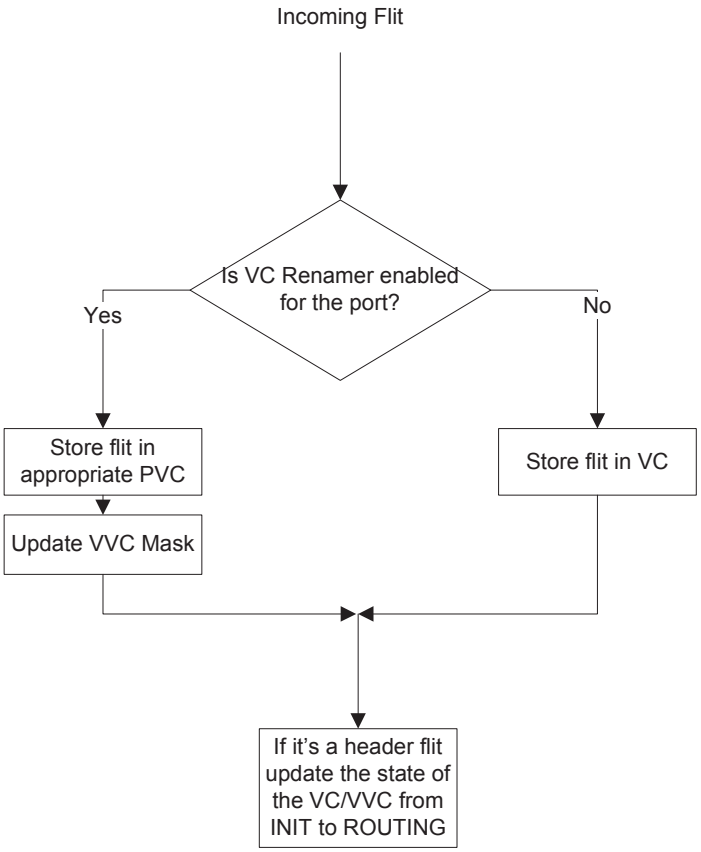


Figure B.1: Mask-Based Architecture Flow Chart - Incoming Flits: The basic steps of the VC Renamer Algorithm needed when a flit arrives at an input port.

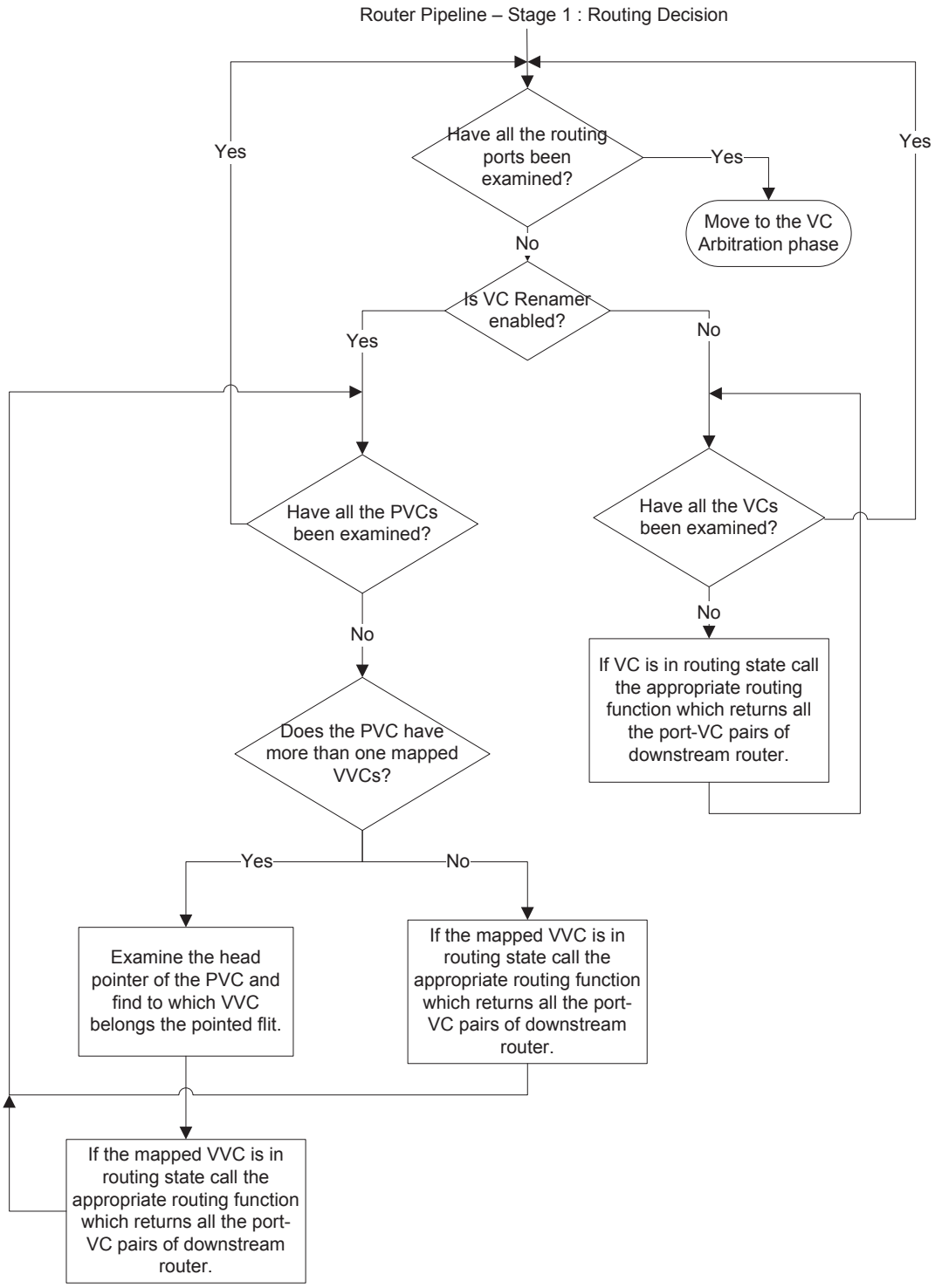


Figure B.2: Mask-Based Architecture Flow Chart - Router Pipeline - Stage 1: Routing Decision: The basic steps of the VC Renamer Algorithm needed during the Routing Computation Stage.

B. VC RENAMER - MB FLOW DIAGRAMS

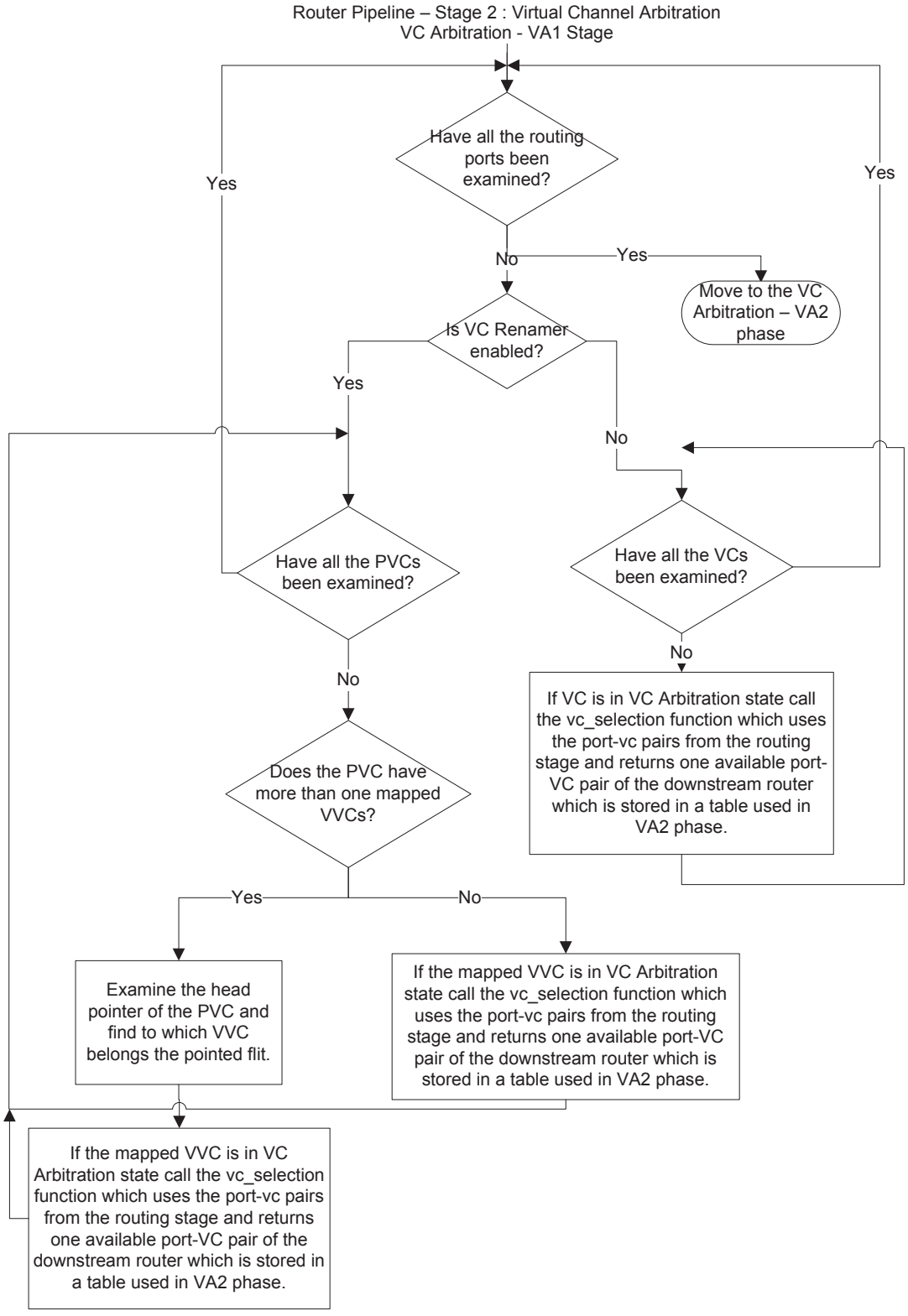


Figure B.3: Mask-Based Architecture Flow Chart - Router Pipeline - Stage 2: VC Arbitration - VA1: the basic steps of the VC Renamer Algorithm needed during the VA1 Stage.

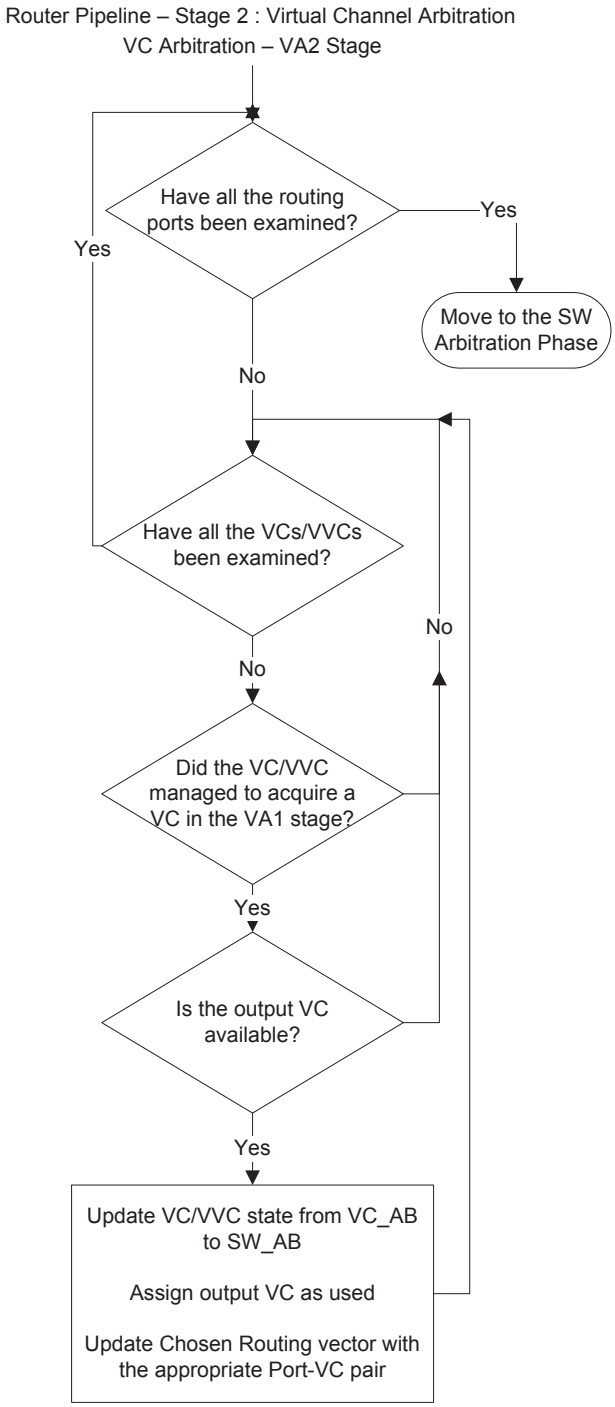


Figure B.4: Mask-Based Architecture Flow Chart - Router Pipeline - Stage 2: VC Arbitration - VA2: the basic steps of the VC Renamer Algorithm needed during the VA2 Stage.

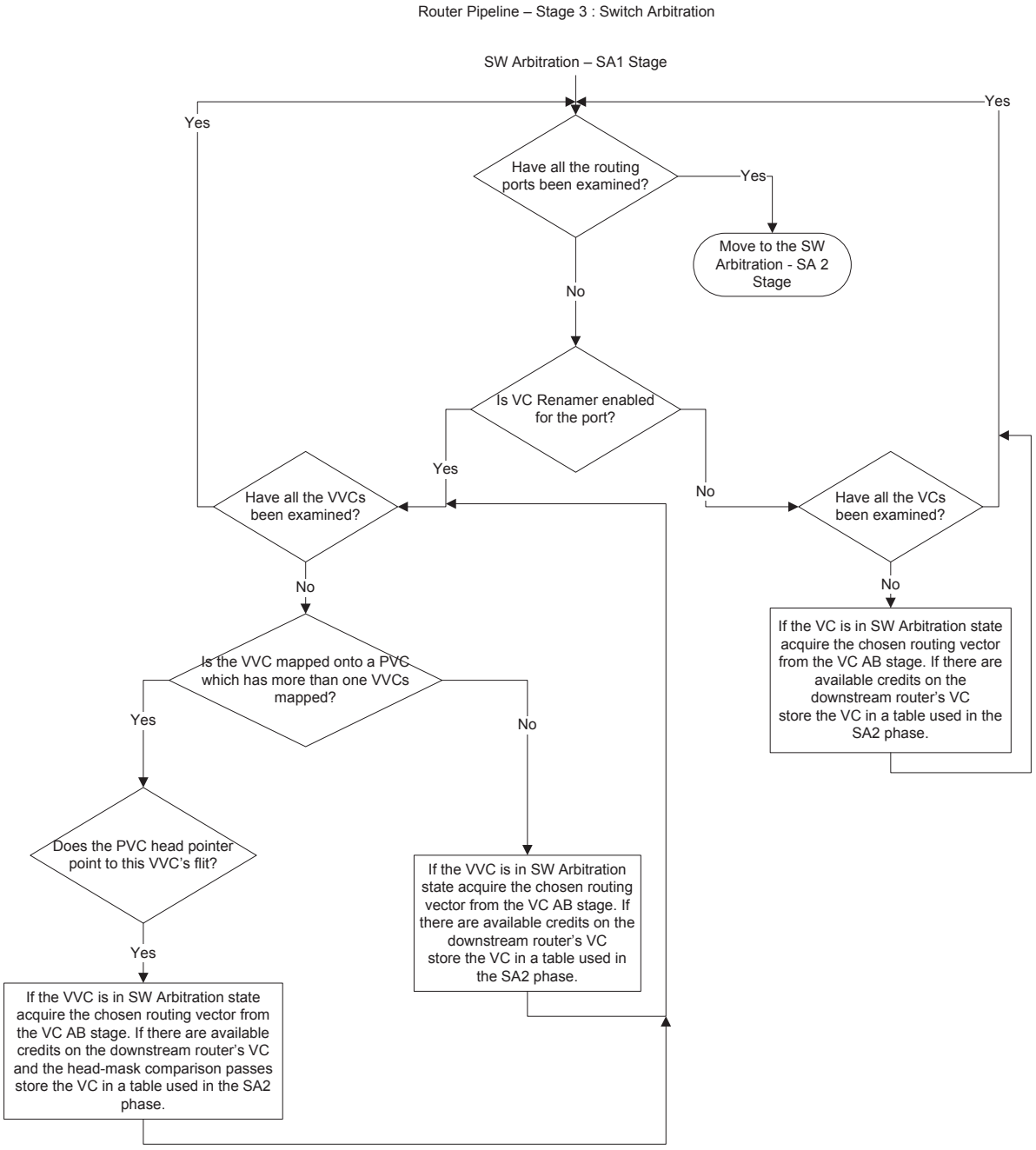


Figure B.5: Mask-Based Architecture Flow Chart - Router Pipeline - Stage 3: Switch Arbitration - SA1: the basic steps of the VC Renamer Algorithm needed during the SA1 Stage.

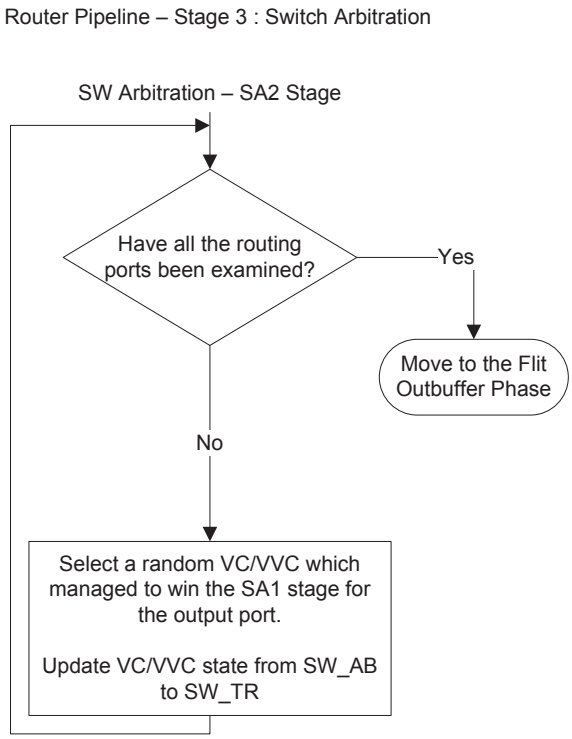


Figure B.6: Mask-Based Architecture Flow Chart - Router Pipeline - Stage 3: Switch Arbitration - SA2: the basic steps of the VC Renamer Algorithm needed during the SA2 Stage.

B. VC RENAMER - MB FLOW DIAGRAMS

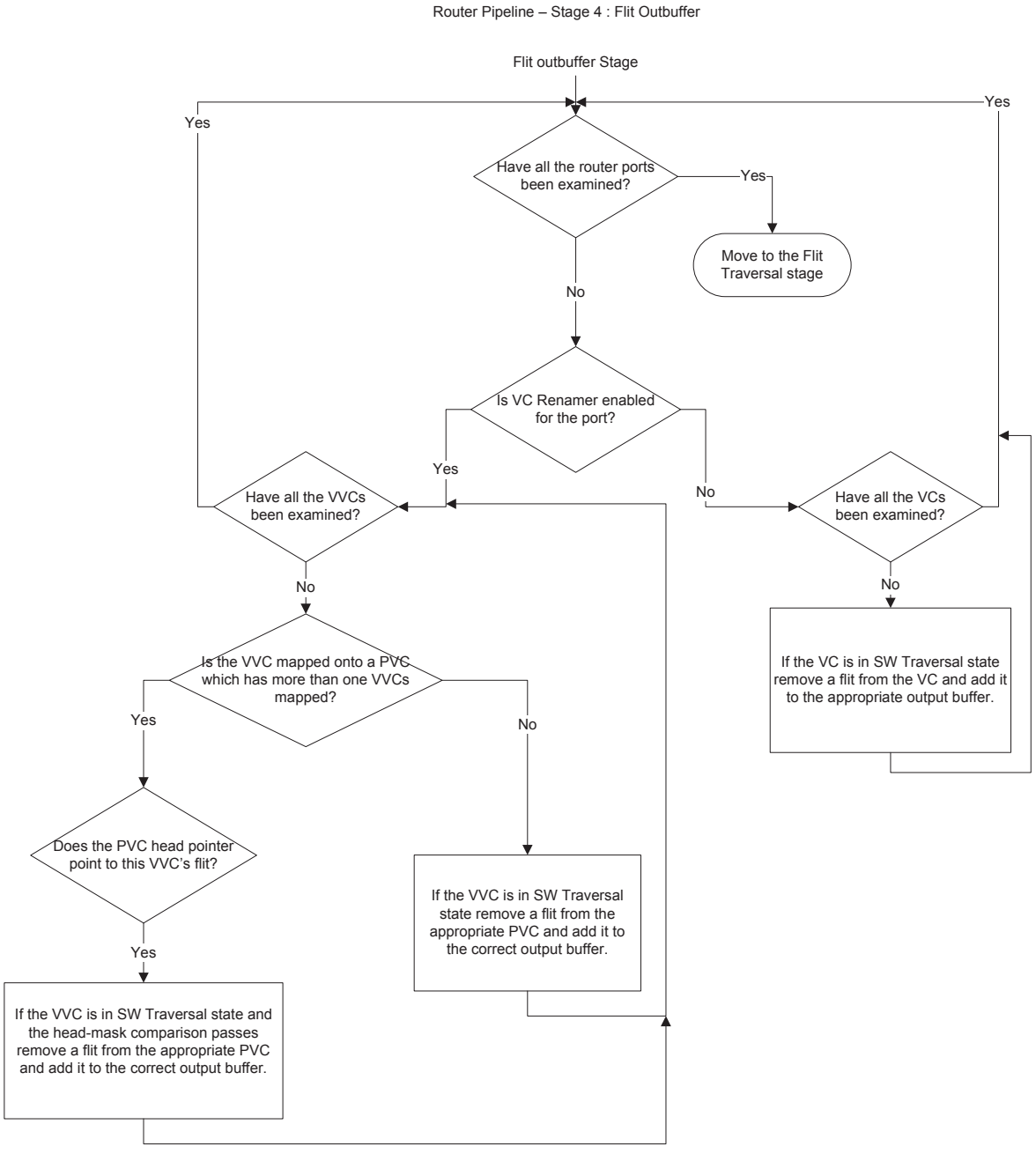


Figure B.7: Mask-Based Architecture Flow Chart - Router Pipeline - Stage 4: Flit Outbuffer: the basic steps of the VC Renamer Algorithm needed when a flit is selected to depart from an input port

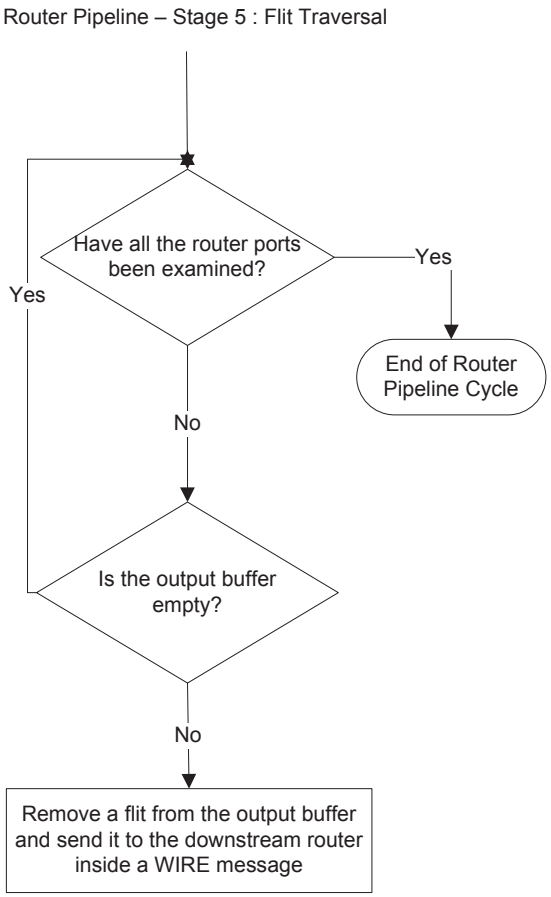


Figure B.8: Mask-Based Architecture Flow Chart - Router Pipeline - Stage 5: Flit Traversal: the basic steps of the VC Renamer Algorithm needed when a flit travels towards the downstream router

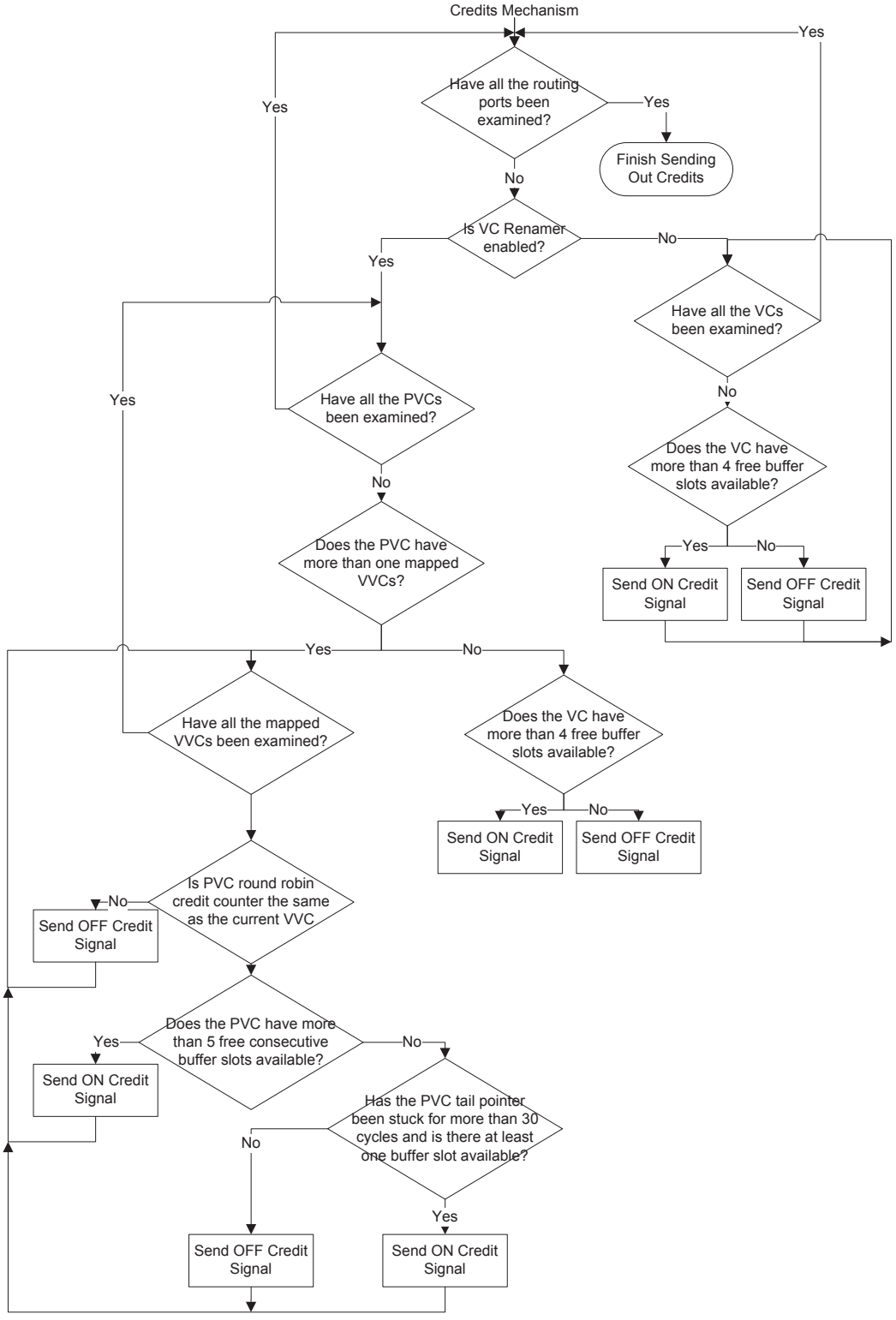


Figure B.9: Mask-Based Architecture Flow Chart - Credit Mechanism: the basic steps of the VC Renamer Algorithm needed when sending out ON-OFF credit messages

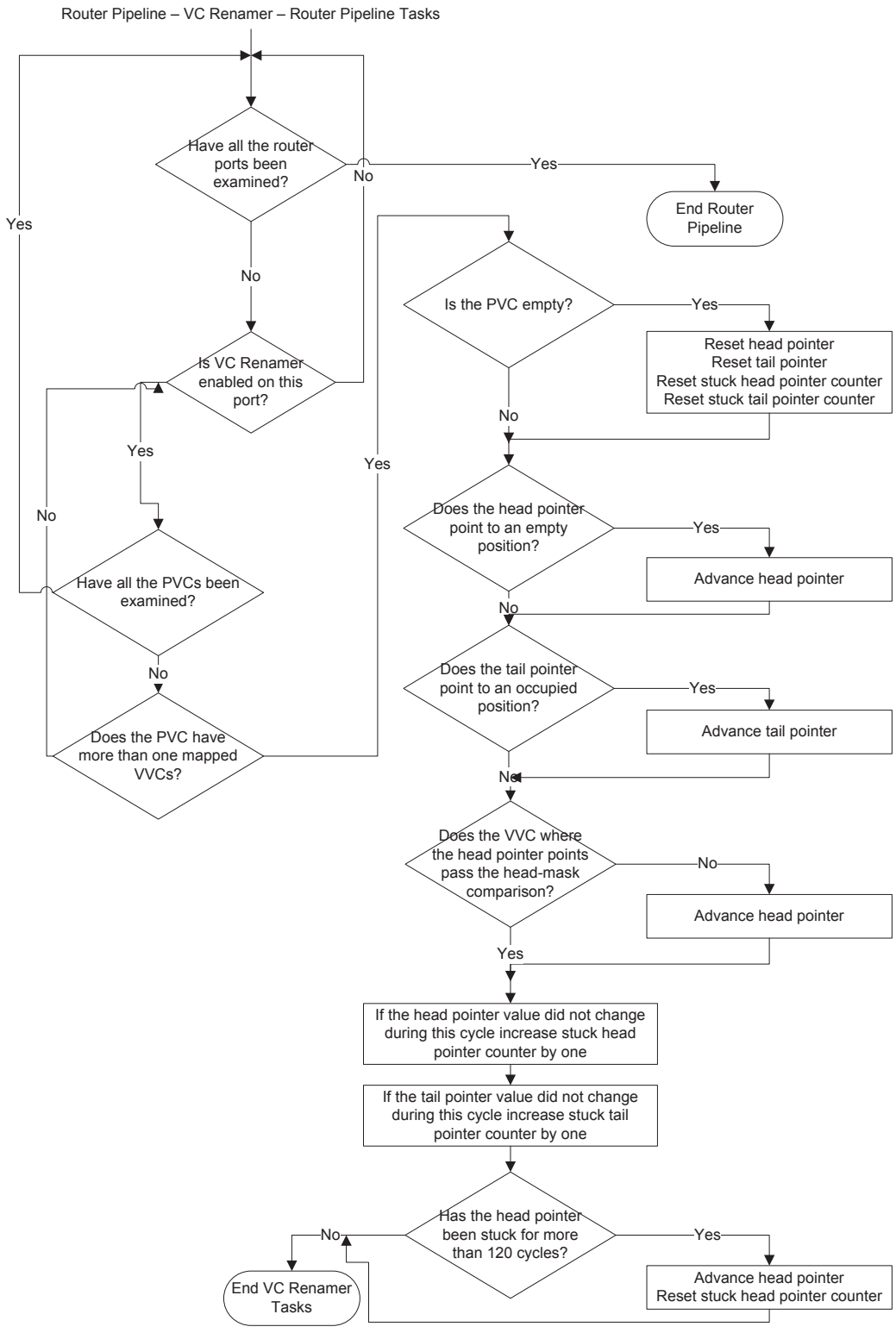


Figure B.10: Mask-Based Architecture Flow Chart - End of router pipeline tasks: the basic steps of the VC Renamer Algorithm needed when the router pipeline ends

Appendix C

VC Renamer - LLB Flow Diagrams

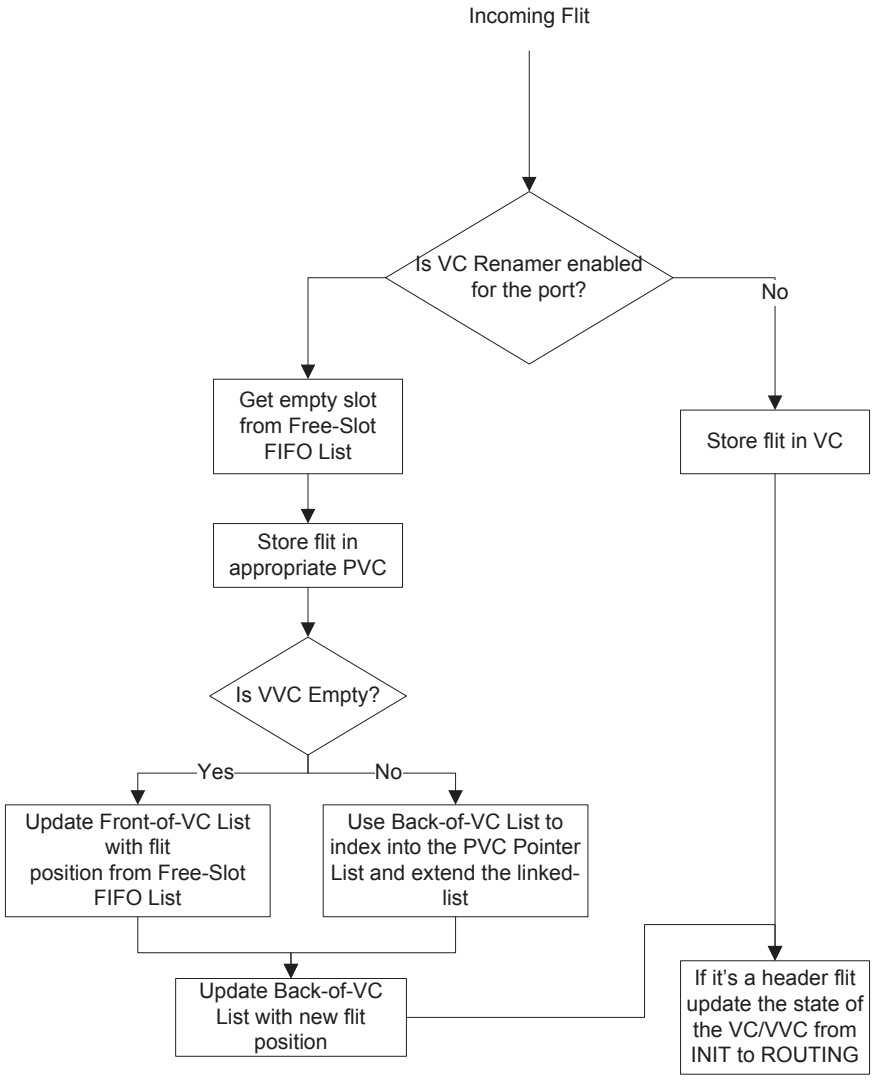


Figure C.1: Linked-List-Based Architecture Flow Chart - Incoming Flits: The basic steps of the VC Renamer Algorithm needed when a flit arrives at an input port.

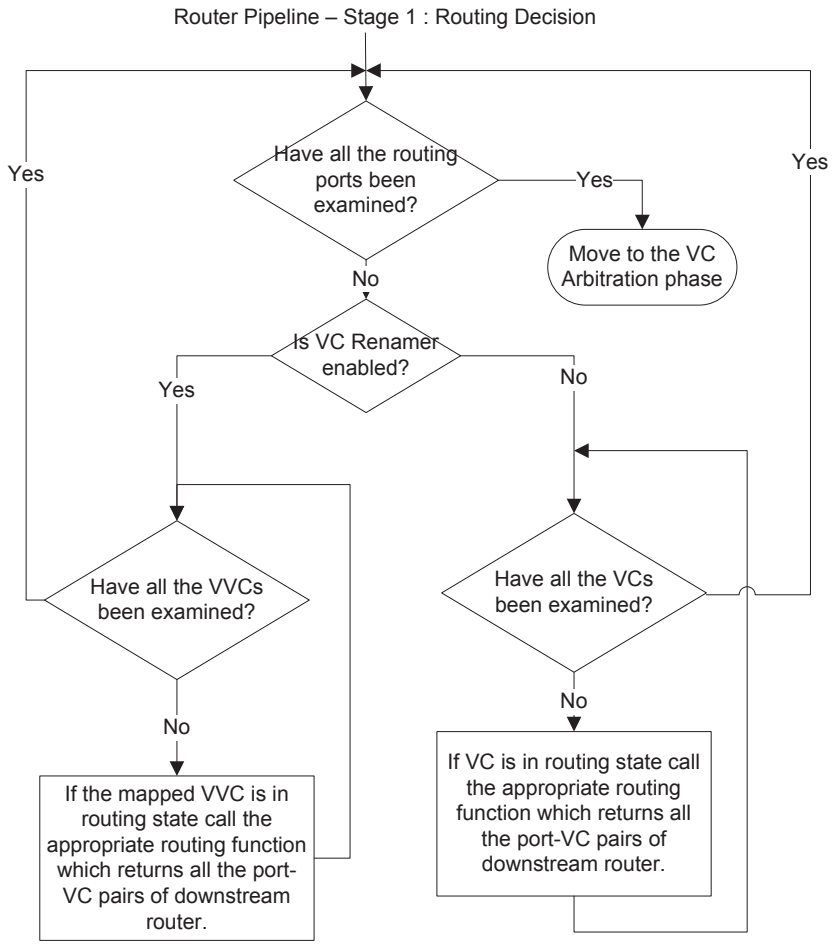


Figure C.2: Linked-List-Based Architecture Flow Chart - Router Pipeline - Stage 1: Routing Decision: The basic steps of the VC Renamer Algorithm needed during the Routing Computation Stage.

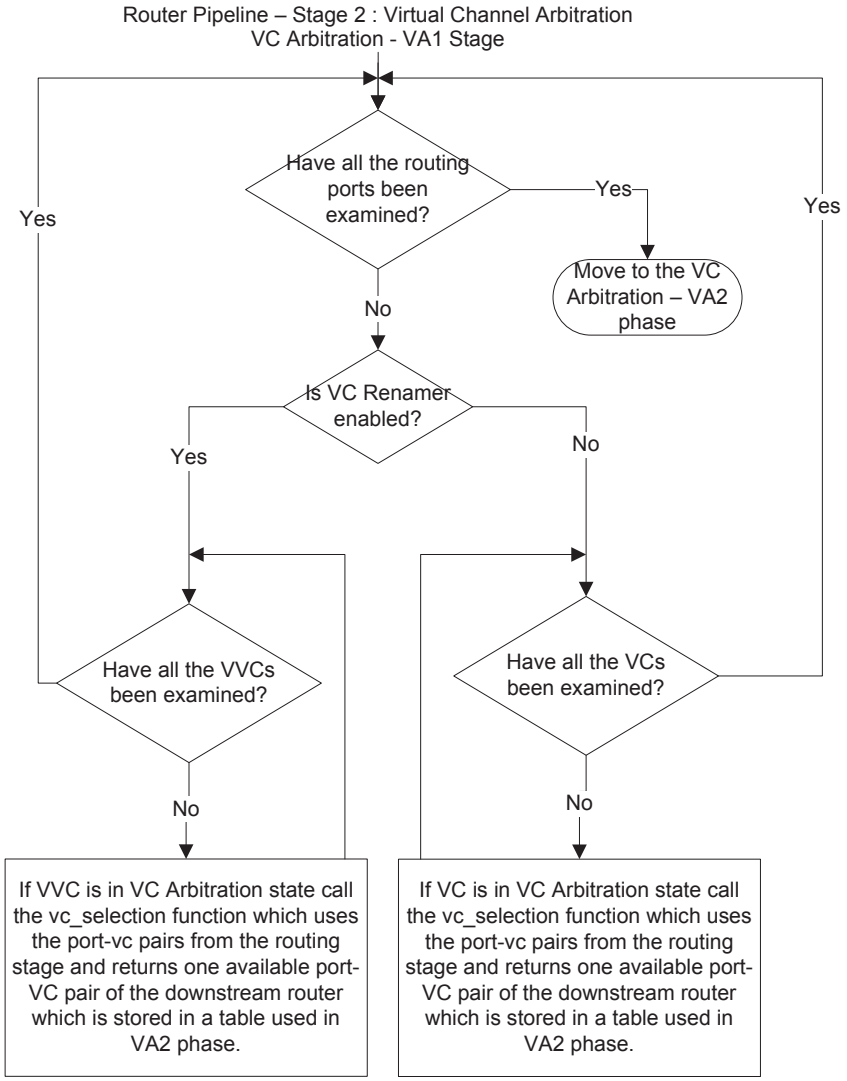


Figure C.3: Linked-List-Based Architecture Flow Chart - Router Pipeline - Stage 2: VC Arbitration - VA1: the basic steps of the VC Renamer Algorithm needed during the VA1 Stage.

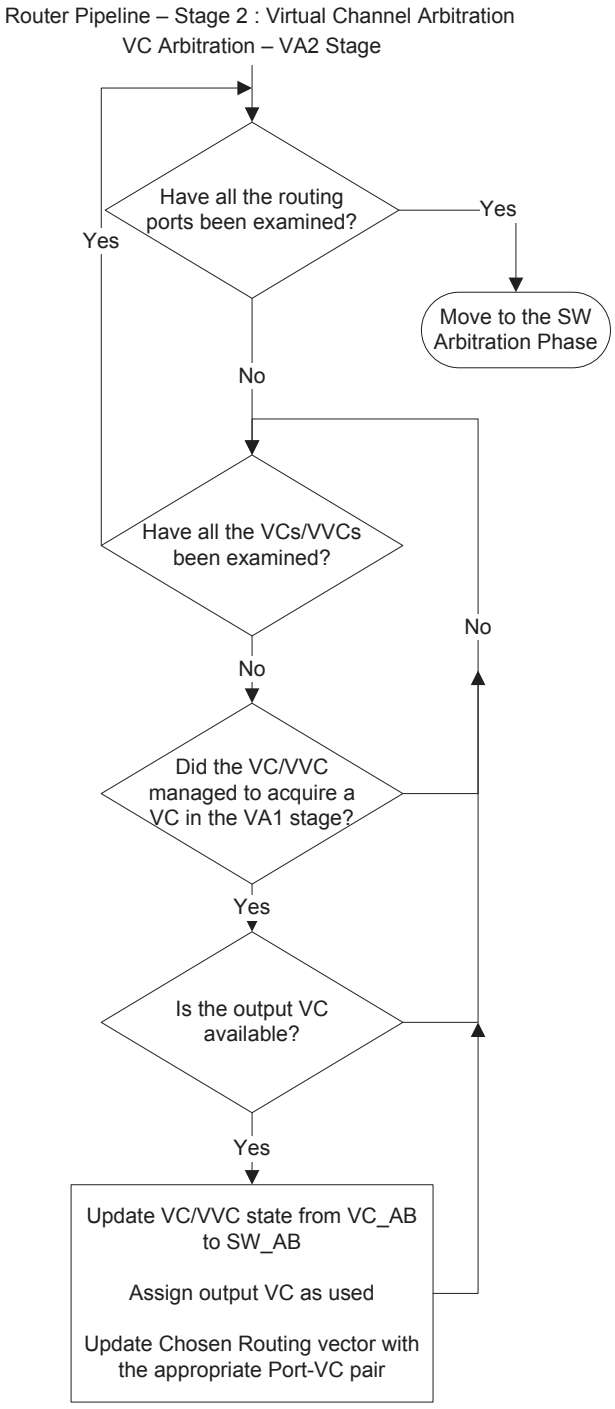


Figure C.4: Linked-List-Based Architecture Flow Chart - Router Pipeline - Stage 2: VC Arbitration - VA2: the basic steps of the VC Renamer Algorithm needed during the VA2 Stage.

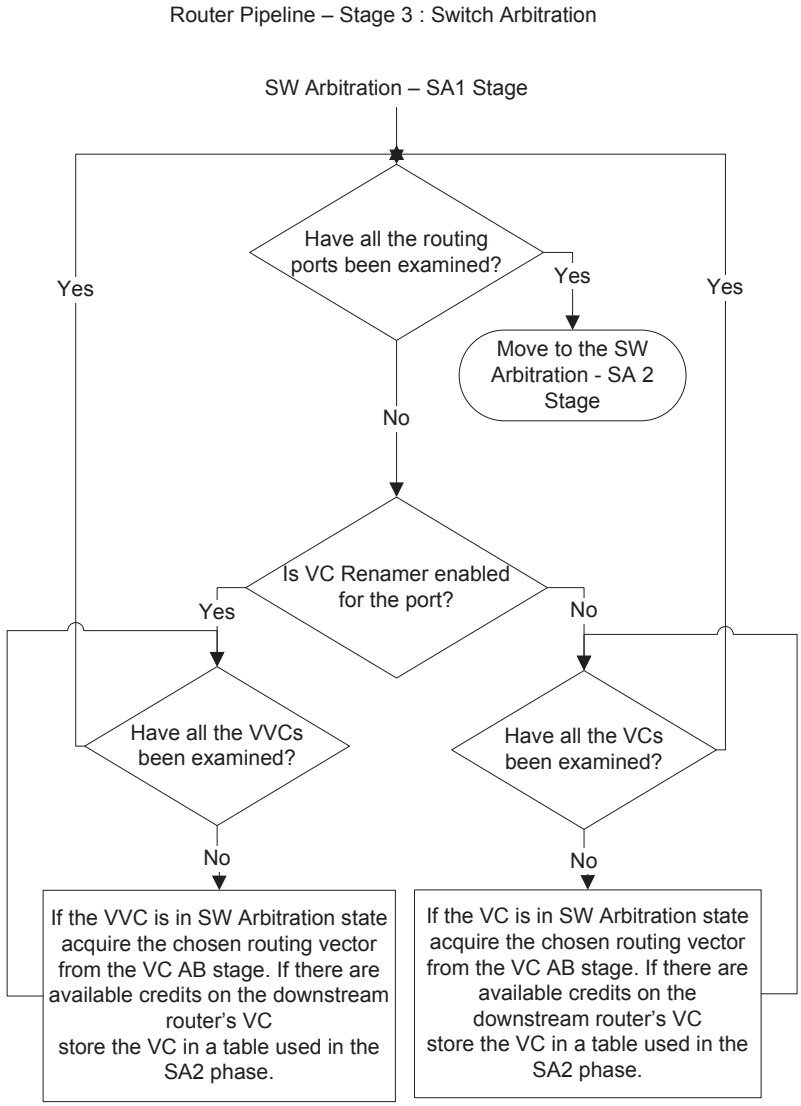


Figure C.5: Linked-List-Based Architecture Flow Chart - Router Pipeline - Stage 3: Switch Arbitration - SA1: the basic steps of the VC Renamer Algorithm needed during the SA1 Stage.

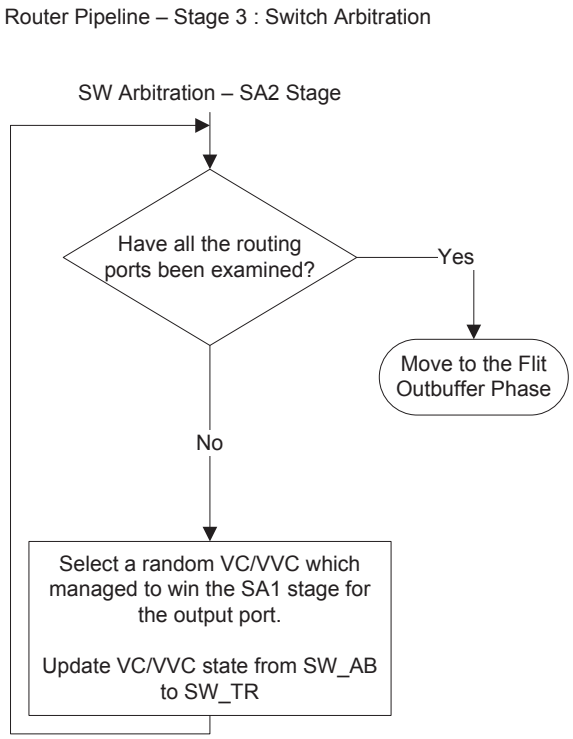


Figure C.6: Linked-List-Based Architecture Flow Chart - Router Pipeline - Stage 3: Switch Arbitration - SA2: the basic steps of the VC Renamer Algorithm needed during the SA2 Stage.

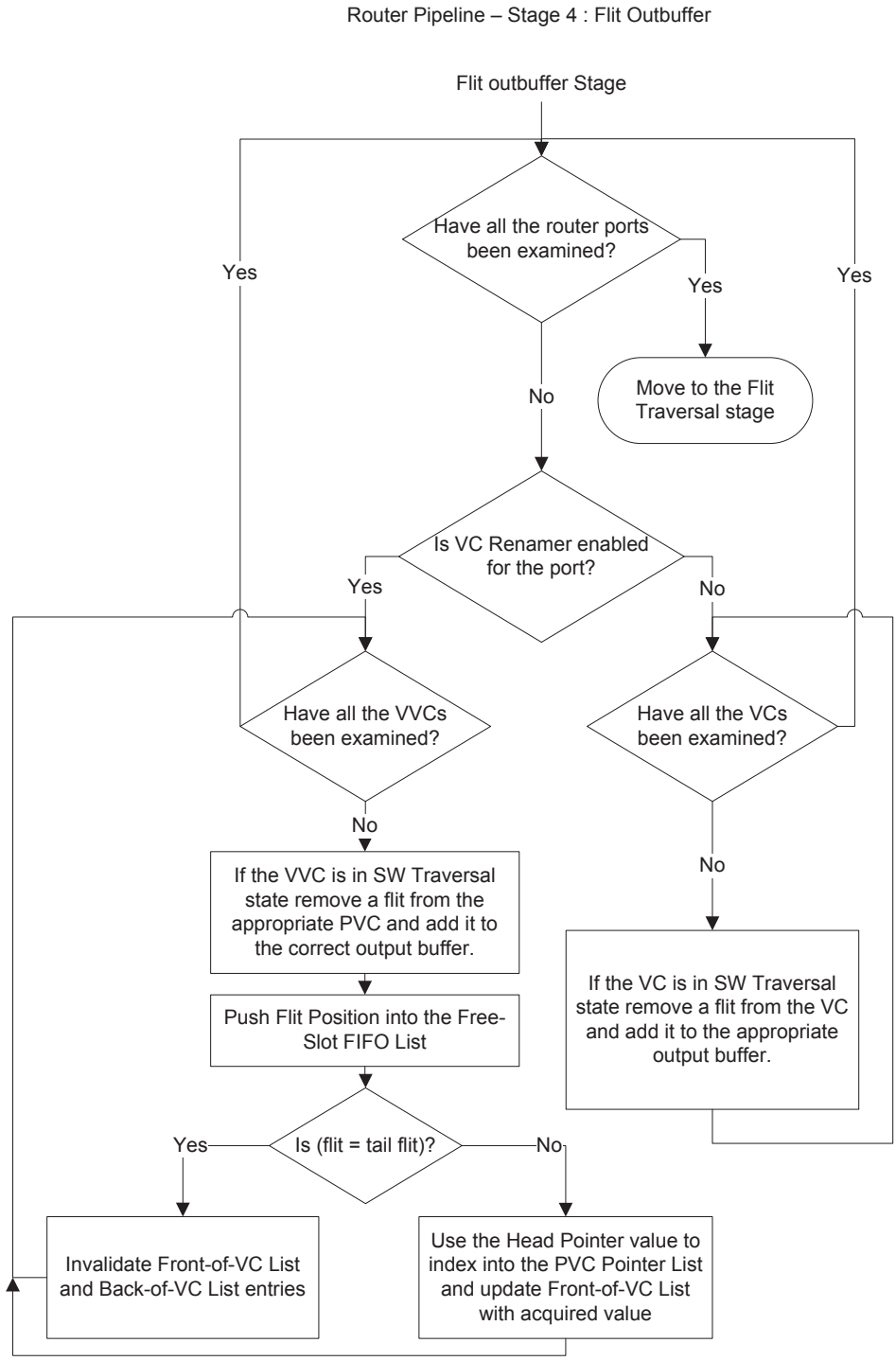


Figure C.7: Linked-List-Based Architecture Flow Chart - Router Pipeline - Stage 4: Flit Outbuffer: the basic steps of the VC Renamer Algorithm needed when a flit is selected to depart from an input port

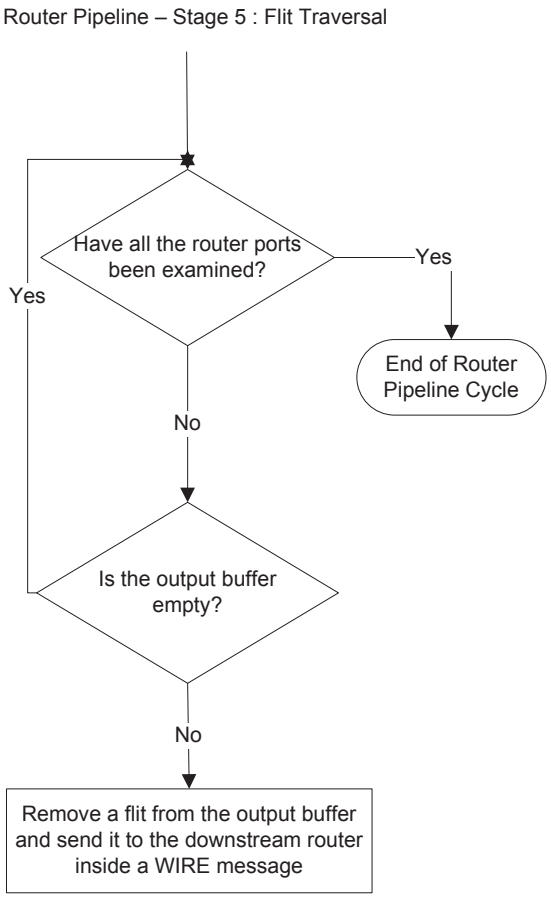


Figure C.8: Linked-List-Based Architecture Flow Chart - Router Pipeline - Stage 5: Flit Traversal: the basic steps of the VC Renamer Algorithm needed when a flit travels towards the downstream router

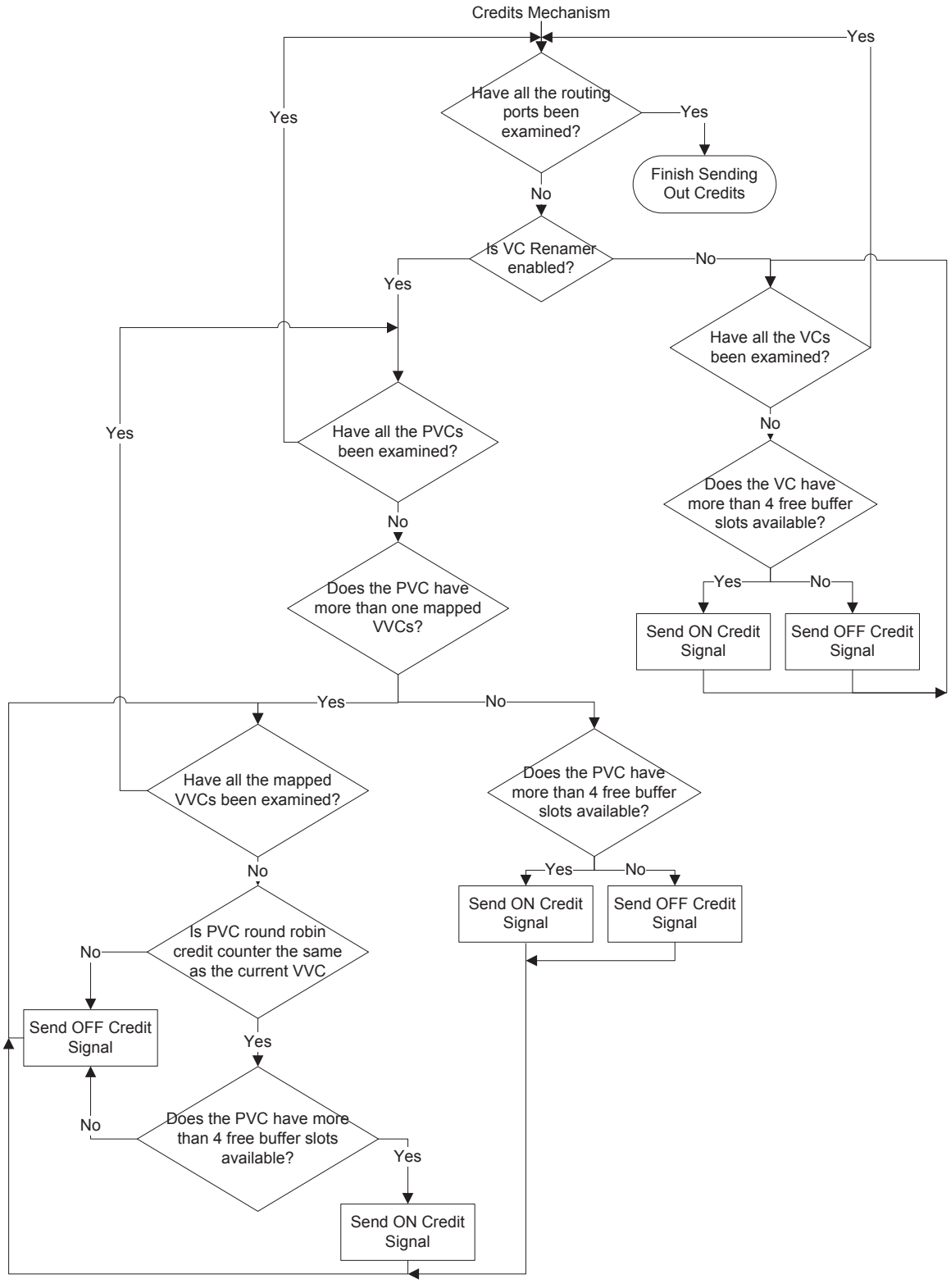


Figure C.9: Linked-List-Based Architecture Flow Chart - Credit Mechanism: the basic steps of the VC Renamer Algorithm needed when sending out ON-OFF credit messages

Bibliography

[1] N.V ijaykrishnan, T. Theocharides, Gregory M. Link and M.J. Irwin. "Networks on Chip (NoC): Interconnects of Next Generation Systems on Chip". In *Advances In Computers*, pages 35 – 89, 2005. vii, 5

[2] L. Benini and G. De Micheli. "Networks on chips: a new SoC paradigm". *Computer*, Volume 35, pages 70 – 78, jan 2002. 1

[3] W.J. Dally and B. Towles. "Route packets, not wires: on-chip interconnection networks". In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 684 – 689, 2001. 1

[4] A. Jantsch S. Kumar A. Postula J. Oberg M.Millberg A. Hemani and D. Lindqvist. "Network on chip: An architecture forbillion transistor era". In *Proceedings of the IEEE NorChip Conference*, 2000. 1

[5] P. Guerrier and A. Greiner. "A generic architecture for on-chip packet-switched interconnections". In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 250–256, 2000. 1

[6] S. Yalamanchili J. Duato and Ni Lionel. *"Interconnection Networks: An Engineering Approach"*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002. 1

[7] N. Agarwal, Li-Shiuan Peh, and N.K. Jha. "In-Network Snoop Ordering (INSO): Snoopy coherence on unordered interconnects". In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 67 –78, feb. 2009. 1

[8] S. Borkar. "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation". *IEEE Micro, 2005*, 25(6):10–16, nov.-dec. 2005. 2

[9] S.R. Nassif, N. Mehta, and Yu Cao. "A resilience roadmap". In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 1011–1016, march 2010. 2

[10] F.J. Sparacio D.W. Anderson and R.M. Tomasulo. "The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling". *IBM Journal of Research and Development*, 1967. 3

[11] John Hennessy and David Patterson. *"Computer Architecture - A Quantitative Approach"*. Morgan Kaufmann, 2003. 3

[12] M. Hayenga, N.E. Jerger, and M. Lipasti. "SCARAB: A single cycle adaptive routing and bufferless network". In *Proceedings of the International Symposium on Microarchitecture*, pages 244–254, dec. 2009. 16

[13] C. Fallin, C. Craik, and O. Mutlu. "CHIPPER: A low-complexity bufferless deflection router". In *Proceedings of the High Performance Computer Architecture Symposium*, pages 144–155, feb. 2011. 16

BIBLIOGRAPHY

[14] Thomas and Mutlu Onur Moscibroda. "A case for bufferless routing in on-chip networks". In *Proceedings of the International Symposium on Computer Architecture*, pages 196–207, 2009. 16

[15] G. Michelogiannakis, D. Sanchez, W.J. Dally, and C. Kozyrakis. "Evaluating Bufferless Flow Control for On-chip Networks". In *Proceedings of the International Symposium on Networks-on-Chip*, pages 9–16, may 2010. 17

[16] M.A. Al Faruque and J. Henkel. "Minimizing Virtual Channel Buffer for Routers in On-chip Communication Architectures". In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 1238–1243, march 2008. 17

[17] Young Hoon Kang, Taek-Jun Kwon, and J. Draper. "Dynamic packet fragmentation for increased virtual channel utilization in on-chip routers". In *Proceedings of the International Symposium on Networks-on-Chip*, pages 250–255, may 2009. 17

[18] Keun Sup Shim, Myong Hyon Cho, M. Kinsy, T. Wen, M. Lis, G.E. Suh, and S. Devadas. "Static virtual channel allocation in oblivious routing. In *Proceedings of the International Symposium on Networks-on-Chip"*, pages 38–43, may 2009. 17

[19] Mingche Lai, Zhiying Wang, Lei Gao, Hongyi Lu, and Kui Dai. "A dynamically-allocated virtual channel architecture with congestion awareness for on-chip routers". In *Proceedings of the Design Automation Conference*, pages 630–633, june 2008. 17, 19

[20] Yi Xu, Bo Zhao, Youtao Zhang, and Jun Yang. "Simple virtual channel allocation for high throughput and high frequency on-chip routers". In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 1–11, jan. 2010. 17

[21] R. Dobkin, R. Ginosar, and I. Cidon. "QNoC Asynchronous Router with Dynamic Virtual Channel Allocation". In *Proceedings of the International Symposium on Networks-on-Chip*, page 218, may 2007. 17

[22] C.A. Nicopoulos, Dongkook Park, Jongman Kim, N. Vijaykrishnan, M.S. Yousif, and C.R. Das. "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers". In *Proceedings of the International Symposium on Microarchitecture*, pages 333–346, dec. 2006. 17, 19

[23] R.S. Ramanujam, V. Soteriou, B. Lin, and Li-Shiuan Peh. "Design of a High-Throughput Distributed Shared-Buffer NoC Router". In *Proceedings of the International Symposium on Networks-on-Chip*, pages 69–78, may 2010. 17

[24] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw. "A highly resilient routing algorithm for fault-tolerant NoCs". In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 21–26, april 2009. 18

[25] Zhen Zhang, A. Greiner, and S. Taktak. "A reconfigurable routing algorithm for a fault-tolerant 2D-Mesh Network-on-Chip". In *Proceedings of the Design Automation Conference*, pages 441–446, june 2008. 18

[26] A. Kohler and M. Radetzki. "Fault-tolerant architecture and deflection routing for degradable NoC switches". In *Proceedings of the International Symposium on Networks-on-Chip*, pages 22–31, may 2009. 18

[27] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato. "Addressing Manufacturing Challenges with Cost-Efficient Fault Tolerant Routing". In *Proceedings of the International Symposium on Networks-on-Chip*, pages 25–32, may 2010. 18

[28] Young Hoon Kang, Taek-Jun Kwon, and J. Draper. "Fault-Tolerant Flow Control in On-chip Networks". In *Proceedings of the International Symposium on Networks-on-Chip*, pages 79–86, may 2010. 18

[29] A. Ejlali, B.M. Al-Hashimi, P. Rosinger, and S.G. Miremadi. "Joint Consideration of Fault-Tolerance, Energy-Efficiency and Performance in On-Chip Networks". In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 1–6, april 2007. 18

[30] R. Marculescu. "Networks-on-chip: the quest for on-chip fault-tolerant communication". In *Proceedings of the Annual Symposium on VLSI*, pages 8–12, feb. 2003. 18

[31] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C.R. Das. "Exploring Fault-Tolerant Network-on-Chip Architectures". In *Proceedings of the Internationa Conference on Dependable Systems and Networks*, pages 93–104, june 2006. 18

[32] D. Fick, A. DeOrio, Jin Hu, V. Bertacco, D. Blaauw, and D. Sylvester. "Vicis: A reliable network for unreliable silicon". In *Proceedings of the Design Automation Conference*, pages 812–817, july 2009. 18

[33] M. H. Neishaburi and Zeljko Zilic. "Reliability aware NoC router architecture using input channel buffer sharing". In *Proceedings of the Great Lakes symposium on VLSI*, pages 511–516, 2009. 18

[34] Mehdi Modarressi, Hamid Sarbazi-Azad, and Arash Tavakkol. "An efficient dynamically reconfigurable on-chip network architecture". In *Proceedings of the Design Automation Conference*, pages 166–169, june 2010. 19

[35] Al Faruque, T. Ebi, and J. Henkel. "Configurable links for runtime adaptive on-chip communication". In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 256–261, april 2009. 19

[36] Y. Tamir and G. L. Frazier. "High-performance multi-queue buffers for VLSI communications switches". In *Proceedings of the International Symposium on Computer architecture*, ISCA '88, pages 343–354, 1988. 19

[37] J. Park, B.W. O'Krafka, S. Vassiliadis, and J. Delgado-Frias. "Design and evaluation of a DAMQ multiprocessor network with self-compacting buffers". In *Proceedings of the International Conference on Supercomputing*, pages 713 –722, nov 1994. 19

[38] N. Ni, M. Pirvu, and L. Bhuyan. "Circular buffered switch design with wormhole routing and virtual channels". In *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors*, pages 466 –473, oct 1998. 19

[39] Yungho Choi and Timothy Mark Pinkston. "Evaluation of queue designs for true fully adaptive routers". 64, pages 606–616, May 2004. 19

[40] K. Sankaralingam, R. Nagarajan, R. Mcdonald, R. Desikan, S. Drolia, M.S. Govindan, P. Gratz, D. Gulati, H. Hanson, Changkyu Kim, H. Liu, N. Ranganathan, S. Sethumadhavan, S. Sharif, P. Shivakumar, S.W. Keckler, and D. Burger. "Distributed Microarchitectural Protocols in the TRIPS Prototype Processor". In *Proceedings of the International Symposium on Microarchitecture*, pages 480–491, dec. 2006. 57, 61

[41] J.L. Henning. "SPEC CPU2000: measuring CPU performance in the New Millennium". *Computer*, 33(7):28–35, jul 2000. 57

[42] A. Agarwal, D. Blaauw, and V. Zolotov. "Statistical timing analysis for intra-die process variations with spatial correlations". In *Proceedings of the International Conference on Computer Aided Design*, pages 900–907, nov. 2003. 57