VIRTUAL CHANNEL ORGANIZATION AND ARBITRATION FOR NETWORK ON CHIP ROUTER ARCHITECTURE

by

Masoud Oveis Gharan

B.S., Isfahan University of Technology, Iran, 1991

M.A.Sc., Ryerson University, Canada, 2011

A dissertation

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2016

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A DISSERTATION

Virtual Channel Organization and Arbitration

For Network on Chip Router Architecture

Doctor of Philosophy, 2016

Masoud Oveis Gharan

Electrical and Computer Engineering

Ryerson University

Abstract

The advent of Multi-Processor Systems-on-Chip (MPSoC) has emphasized the importance of on-chip communication infrastructures. Network on Chip (NoC) has emerged as a state of the art paradigm for efficient on-chip communication. Among the various components employed in NoC routers, Virtual Channel (VC) plays an important role in the performance and hardware requirements of an NoC system. The VC mechanism enables the multiplexing and buffering of several packets to travel over a single physical channel concurrently. VC arbitration (or arbiter) is another critical organization component of a router that has significant impact on the efficiency of an NoC system. Arbiter performs arbitration among the group of VCs that are competing for a single resource (e.g. output-port).

In this dissertation, we propose novel approaches for dynamic VC flow control mechanism and VC arbitration. The first two approaches are based on the adaptivity of VCs in the router input-port that improves the efficiency of NoC system. In both of techniques, the input-port comprises of a centralized buffer whose slots are dynamically allocated to VCs according to a real-time traffic situation. The performance improvement is achieved by utilizing multiple virtual channels with minimal buffer resources. The VC arbitration approach is based on an efficient and fast arbiter that functions upon the index of its input-ports (or VC requests). The architecture of arbiter scales with the Log_2 of the number of inputs where a conventional round robin arbiter scales with the number of inputs. The index based behavior and the architecture of our arbiter leads to lower power consumption and chip area.

This dissertation presents the organizations and micro-architectures of NoC routers. We have employed SystemVerilog at the micro-architectural level design and modeling of NoC components. We employ three CAD platforms namely ModelSim, Quartus (FPGA) and Synopsys (ASIC level) to design, simulate and implement our router based NoCs. The simulation results support the theoretical concepts of our proposed VC organization and arbitration approaches. We have also implemented and conducted simulation and modeling experiments for conventional VC organization and arbitration models. The experimental results verify the efficiency of our proposed models in terms of power, area and performance in different NoC configurations.

Acknowledgment

Hereby I truly thank all Ryerson University administrators and staffs who helped and support me in financial and registration matters to do the best for my education. I really appreciate the teaching efforts of the professors who taught me during these five academic years. Here I would like to present my especial thanks to Prof. Gul N. Khan for being an exceptional advisor for my entire graduate career. As advisor and friend, he has managed to perfectly balance encouragement and criticism, and I am in his debt for providing me with the facilities to be a successful researcher in an exciting field. I am astounded by his great breadth of knowledge, and ability to focus on the details. His attention to details and network-on-chip opinion is an invaluable asset, and I am really thankful to be his PhD student.

Above all, I am thankful for the love and support of my family. I am positively blessed to have a wife Maryam, a daughter Ghazal, and a baby boy Erfan as wonderful as mine who have given me an amazing life. Their belief in me was a motivator in everything I did, and their advice and understanding could not be more appreciated.

I owe many thanks to all my friends especially those who are in Micro-system Research Laboratory for their unwavering support and friendship. I love spending time with you guys, some of my best memories have been made here.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

In this chapter, we introduce the approaches of Virtual Channel (VC) organization and arbitration for novel NoC router design. First of all, we try to present some commonly used terms, mechanisms and micro-architectures used in the design of Network on Chip (NoC) systems. We also introduce the problems related to the conventional NoC designs as well as the objectives that are pursued in this dissertation. Overall, we present three approaches related to NoC design. The first two approaches are related to VC organization and presented in Chapters 3 and 4. The third approach is related to data flow arbitration and key to our rapid NoC router architecture presented in Chapter 5.

1.1 Network on Chip a State of the Art Paradigm

Network on Chip (NoC) architecture provides a communications infrastructure for the cores of a multi-core System-on-Chip (SoC). The NoC enables the SoC-cores to communicate among each other concurrently by sending messages asynchronously. NoC structures improve the scalability and power efficiency of complex SoCs as compared to other conventional communication systems. Figure 1.1a illustrates an SoC including some IP cores that are connected through a 3×3 Mesh NoC architecture. The NoC includes a network of routers (switches) that are interconnected by data links.

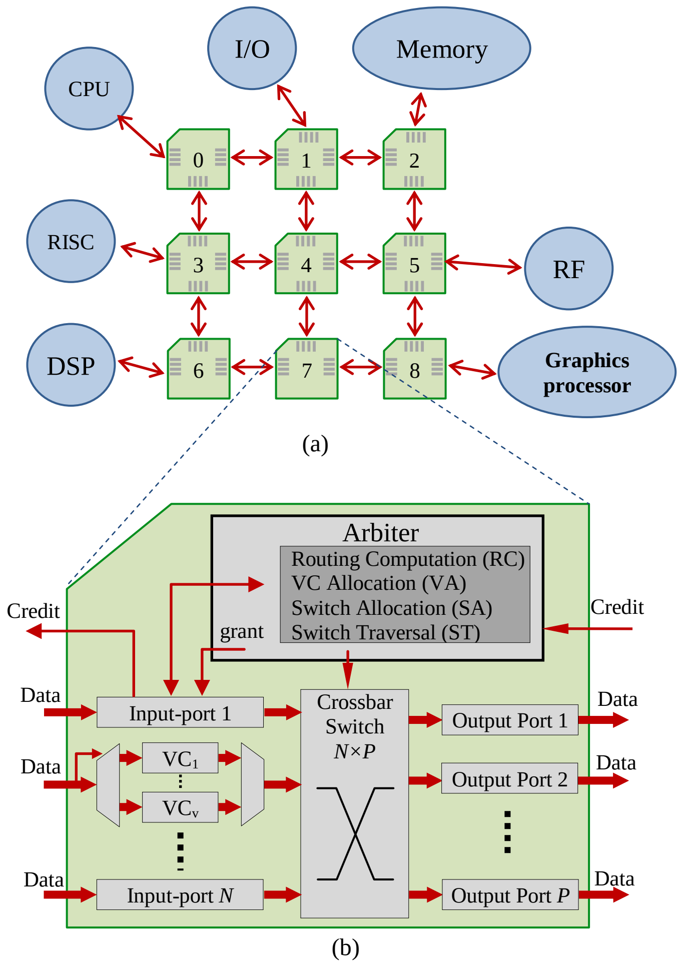


Figure 1.1: a) An SoC with 3×3 Mesh NoC architecture, b) A router architecture with *N* inputs and *P* outputs.

1.2 Wormhole Routing

A most viable communication mechanism employed in NoCs is packet-based wormhole routing [1]. The message in wormhole routing is made of multiple packets where each packet consists of multiple flits. A flit is a basic unit of data that is generally transferred at (the NoC) clock rate. Figure 1.2 illustrates the message structure of a packet-based wormhole flow control. The first flit of a packet is called the header flit and holds the route information of the associated packet. The remainder flits are called body flit where the last flit is known as tail flit. The body and tail flits contain data and may also contain two pieces of information: tail state and VC identification as shown in Figure 1.2. When the header flit of a packet passes through a route made of routers, the route path is reserved for that packet. The route path remains reserved until all the packet flits pass through it.

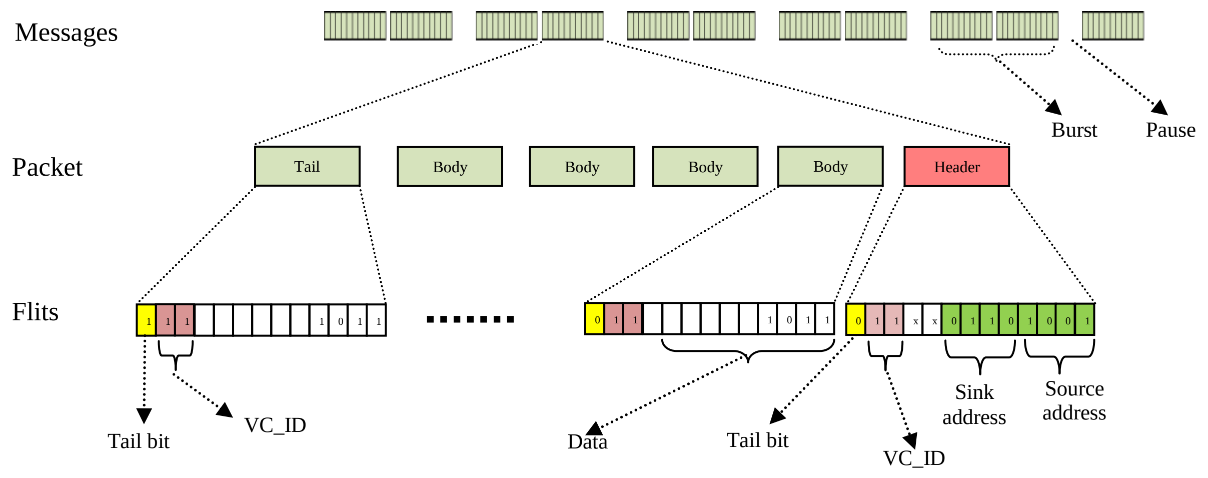


Figure 1.2: A Typical Wormhole Packet Structure in NoC.

For example, consider a 2D mesh NoC wormhole communication situation depicted in Figure 1.3, where the source, S1 sends a packet consisting of 4 flits (Hf, Bf1, Bf2 and Tf) to the destination D9. Assume that the data (packets) movement follows XY routing methodology where the packets first move in X direction to reach to the Y dimension of their destinations, then they move to Y direction to reach their destination. In a basic NoC communication (no VC), passing the header flit, Hf through each router leads the router's input and output ports to be reserved, where no other packets can pass through those ports. After the passage of tail flit, Tf through each router leads the release of that port. During the packet transfer, the flits are temporarily kept in the input (or output) port buffers of the routers in a First-In First-Out (FIFO) fashion. One important feature of the data flow in the case of Figure 1.3 is that the route-path between S1 and D9 is reserved when the flits move through it. Therefore, the flow of data moves without any blocking. Such kind of data flow does not always provide optimum performance. Other messages of other sources where routes or part of their routes are shared with the reserved route have to wait until the route becomes free. These waiting conditions continue even when there is no communication through the reserved route. Sometimes, the reserved route is sometimes idle and will block other packets to pass through it. This kind of data flow incurs higher latency and lower utilization of shared NoC resources. One way of alleviating this problem and improve NoC throughput is by utilizing Virtual Channel (VC). To facilitate creating multiple VCs per physical channel, the messages are allocated in units of flits. The term "channel" or "physical channel" in this dissertation refers to a data link that connects the output-port and input-port of two interconnecting routers. The critical role of VC mechanism and architecture in the efficiency of NoC has encouraged us to present two novel VC organizations in Chapters 3 and 4. To understand the VC organization, we need to explore the structure and mechanism of a NoC router.

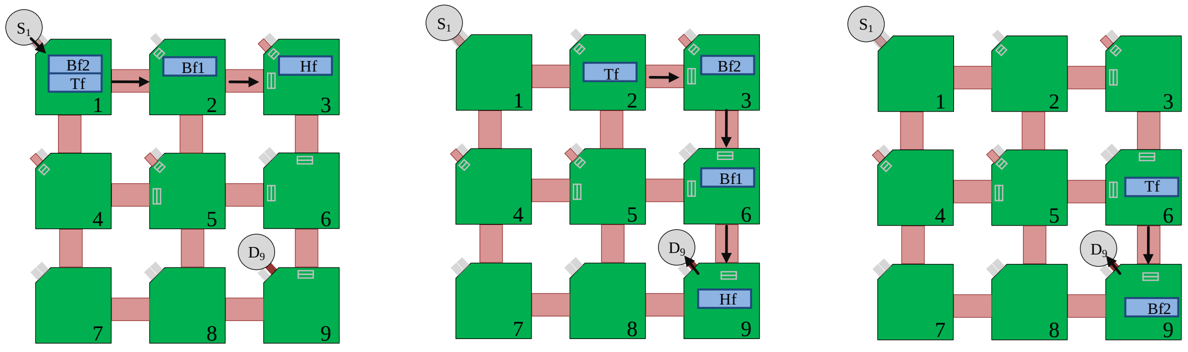
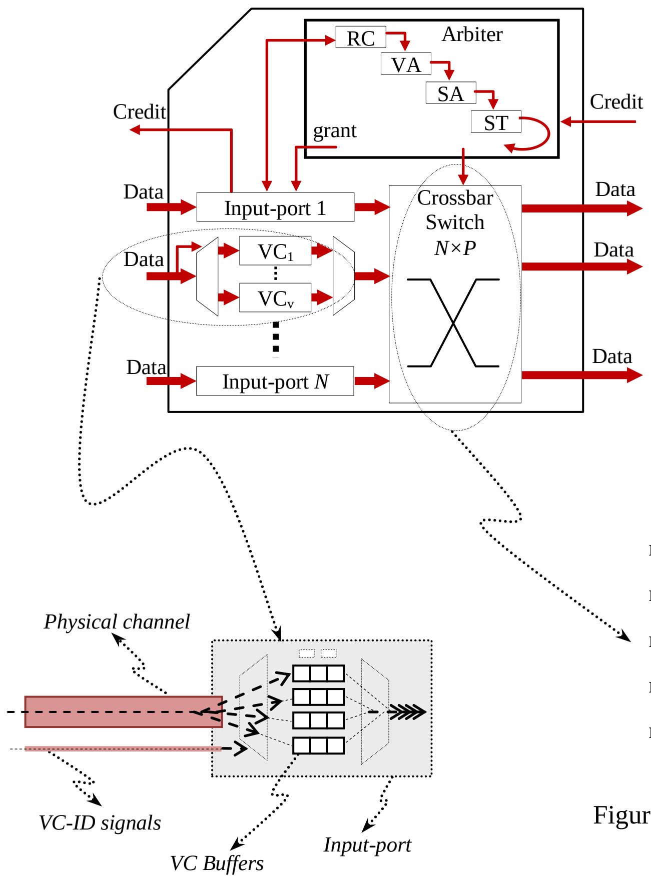


Figure 1.3: Source, S1 sends a packet consisting of 4 flits (Hf, Bf1, Bf2 and Tf) to destination D9.

1.3 NoC Router Architecture

An NoC router accepts packets from the source core (or other router modules) and delivers them to the sink/destination core (or other router modules). The traditional micro-architecture of a router consists of input and output ports, an arbiter, and a crossbar switch as illustrated in Figure 1.1b [1]. The input and output ports can be simple data buses that connect a router to its channels, but at least one of them should consist of a circuit to perform buffering and traversal of the incoming flits. In this dissertation, the input-ports utilize the buffers, and the output-ports are simple data buses. After buffering a flit, the input-port issues a request signal to the arbiter. The arbiter performs arbitration among the potential VC flits that make request to access the crossbar and other shared resources [2]. When a flit wins arbitration and is granted to exit the router, it passes through the crossbar switch. For NoCs utilizing VCs, the structure of router input-port becomes complex. However, it significantly improves the efficiency of NoC. The crossbar switch can be configured to connect any input buffer of the router to any output channel (port), but under the constraints that an input-port is connected to only one output-port. The micro-architecture of crossbar switch is simple as illustrated in the multiplexer-based architecture of Figure 1.4. The structure of a router arbiter can be simple when an NoC does not utilize VC organization. However, the arbiter becomes complex for a VC based router that can have a direct effect on the efficiency of NoC. The critical impact of arbiter on the efficiency of NoC has encouraged us to design a fast and efficient arbiter in Chapter 5.



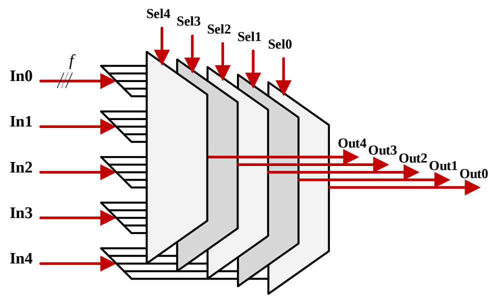


Figure 1.4: A Multiplexer-based Crossbar Switch.

Figure 1.5: Conventional VC Flow Control Communication.

1.4 VC organization

To improve the data flow efficiency in NoCs, each input or output port can utilize VCs to share a physical communication channel by multiple packets. A VC virtually splits a single physical channel to provide two or more virtual paths for the packets to be routed. Consider the case illustrated in Figure 1.3, two packets can reserve and pass through the same route if there are 2 VCs available for each physical channel. In other words, the flits of one packet will interleave with the flits of the other packet over a physical channel by using a rotating flit-by-flit arbitration. The routing of each flit can be guaranteed because the flits belonging to a packet are attached with the VC identification (*VC-ID*) tag at each router. Then these flits become differentiable at the downstream routers. Figure 1.2 illustrates a 2-bit *VC-ID* tag that exists in all the flits of a packet, and these tags are identical when the flits enter a router. Figure 1.5 illustrates the conventional micro-architecture of a VC decoder based on a simple de-multiplexer. The *VC-ID* is connected to the selection port of the de-multiplexer and causes the incoming flit to reside in the associated VC buffer. Basically, the flits of a packet are always stored in the same VC buffer. We assume that a router implements VCs at the input-ports and the input-port storage (buffer) temporarily stores the incoming flits. When a router receives a packet flit, it puts the flit into its input-port buffer, and the flit remains there until the required resources for departure becomes available.

1.4.1 FIFO Architecture

We introduce the architecture and mechanism of data buffering organization (port architecture) in the NoC routers. As mentioned earlier, the data buffering can be in the form of First-In First-Out (FIFO) to keep the flits of a packet in correct order during communication. Two types of FIFO schemes: serial and parallel have been utilized in digital design [3, 4, 5, 6]. The serial FIFO (e.g. shift register) that works on the fall-through principle (or pipeline) has been the initial FIFO type. However, the architectures of conventional FIFOs are constantly being improved. Currently, most of the FIFOs used are of parallel type, which are faster than serial FIFOs [7].

The proposed schematic of Figure 1.6 shows a parallel register-based FIFO. *Write-pointer* and *Read-pointer* are two circular counters that are connected to the selection ports of de-multiplexer and multiplexer. When a write is requested, the *Write-pointer* enables the tail register to store the incoming flit. Then it is incremented to enable the next free slot in the FIFO.

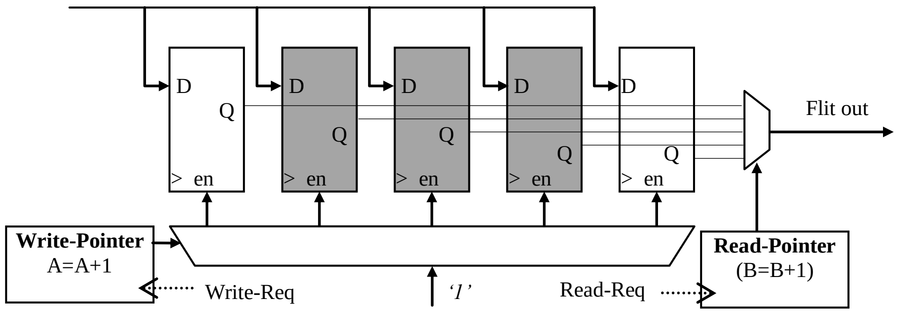


Figure 1.6: Register -based Parallel FIFOs

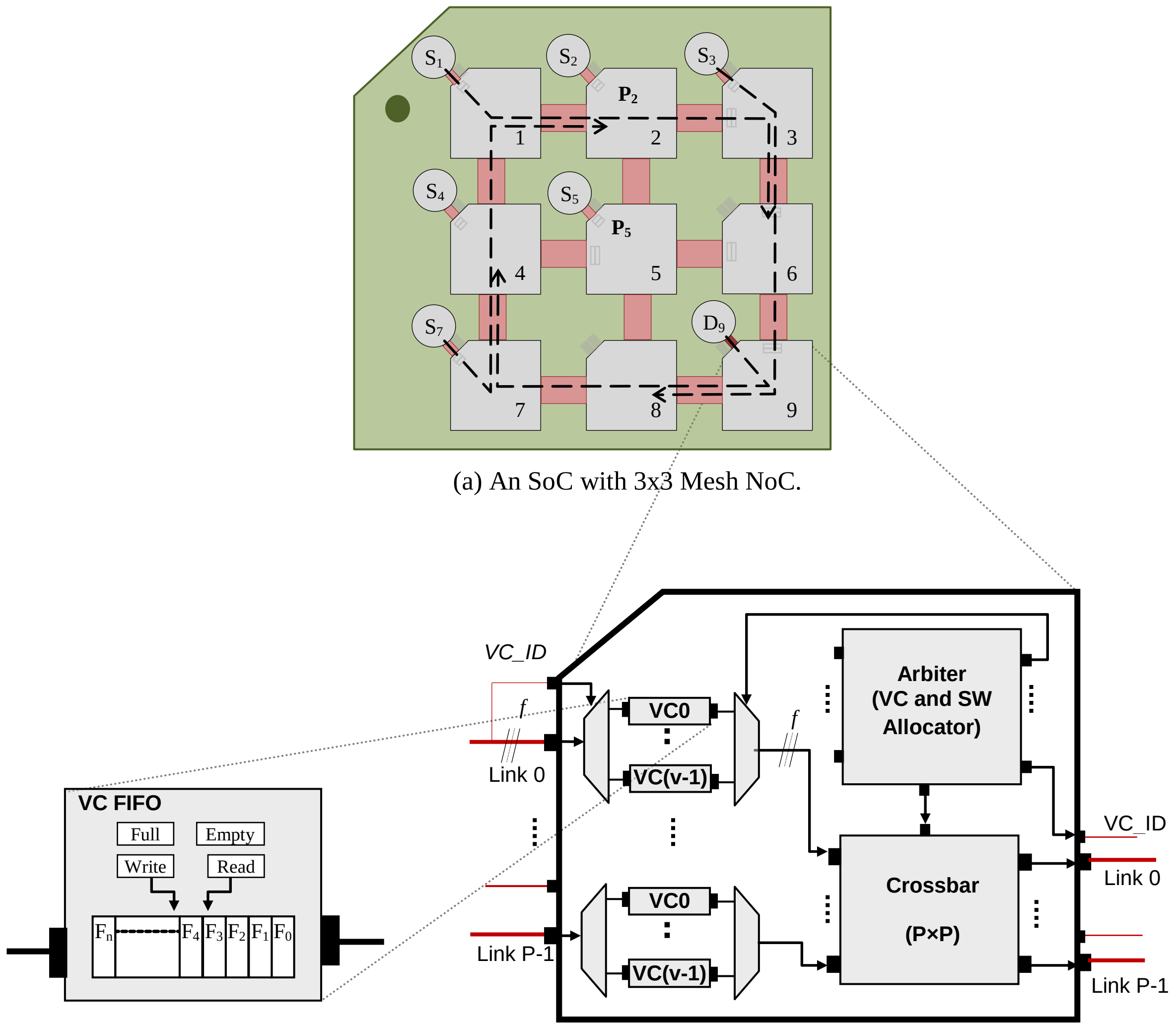
When a read is requested, the *Read-pointer* is incremented to select the output of head slot in the FIFO. Since the pointers increment in a circular manner, the flits enter and exit in a circular manner too. Due to which, the parallel FIFO is also known as a Circular Queue (CQ).

1.4.2 Queue Buffers

In addition to FIFO, another buffering component used in a NoC router is the Queue. A Queue temporarily stores the flit(s) of a packet in a first come first serve (FCFS) manner until the network resources become available. FIFO and Queue terms are sometimes interchangeably referred as an NoC buffer. However, a Queue refers to all types of buffers with the FCFS concept that also contains FIFO buffers. In terms of architecture, FIFO mostly refers to serial or parallel FIFOs as discussed earlier. Serial and parallel FIFO designs are very common in digital design. The concepts discussed in this and previous sections are helpful to clarify the main difference between the VC organizations presented in Chapter 3 and 4. The Chapter 3 approach presents a simple and adaptive multi-FIFO buffer architecture, where the Chapter 4 approach is a small and dynamic multi-queue buffer organization. In the following section, we explain the mechanism and pros and cons of traditional VC organization.

1.4.3 Traditional Wormhole Routing

Wormhole routing is a conventional communication mechanism used in VC based NoC systems. To employ wormhole routing in NoCs, the flit that is part of a packet is buffered at least at the input or output ports of the router [1]. Once the flit of a packet occupies the buffer of a channel, no other packet can access the channel even when the channel buffer is empty. This type of switching flow is prone to contentions and in some cases deadlocks. The contention occurs when the latency of a flit becomes more than the time delay of its location (i.e. router). For example, when a flit is blocked in a router, it should stay in the router until its requested output becomes available. Therefore, the time delay of flit will get higher than the time delay of router that creates a contention situation. Another important issue in the performance of NoCs is the deadlock. Assume wormhole switching and a packet flit is not allowed to pass an element of NoC twice. Then a deadlock will occur when no packet can advance because each packet requires a channel that is already occupied by the other packet. Consider the deadlock situation illustrated in Figure 1.7a. Assume the sources 1, 3, 7 and 9 start sending packet at the same time to the destinations 6, 8, 2 and 4 respectively. The dashed lines in the figure show the route of each packet. The communication becomes deadlock at routers 1, 3, 7 and 9 as each packet requires a channel that is already occupied by the other packet.



(a) An SoC with 3x3 Mesh NoC.

(c) A static VC buffer with parallel FIFO.

(b) A $p \times p$ static v-VC router.

Figure 1.7: SoC (NoC), Router and Queue Architectures.

One of the traditional ways to alleviate contention and remove deadlock is to use the VC mechanism [8]. Consider a configuration where the number of VCs for each channel is equal to the maximum shared packets of that channel. Therefore, as soon as a flit reaches to a router, there will be a free VC facilitating the flit to move to the next router. This means there will be no blocking, and deadlock will not occur in the NoC. Virtual channels are also used to improve message latency and NoC throughput. By allowing messages to share a physical channel, the messages can make progress rather than remain blocked. Moreover, the overall time that a message is blocked at a router and waits for a free channel is lowered. In this way, the sharing of physical channel leads to faster NoC communication and an overall reduction in message latency. In Conventional Virtual Channel (CVC) method, a physical channel support several virtual channels that are multiplexed across the physical channel as shown in Figure 1.5. As depicted in the router of Figures 1.7b, the implementation of VCs needs extra resources i.e. *FIFO* buffer for each VC, *De-multiplexers* and *Multiplexers* for each input-port as well as *VC allocator* and bigger *switch allocator* for the arbiter [1, 9, 10, 11, 12].

1.5 Static and Dynamic VC Organizations

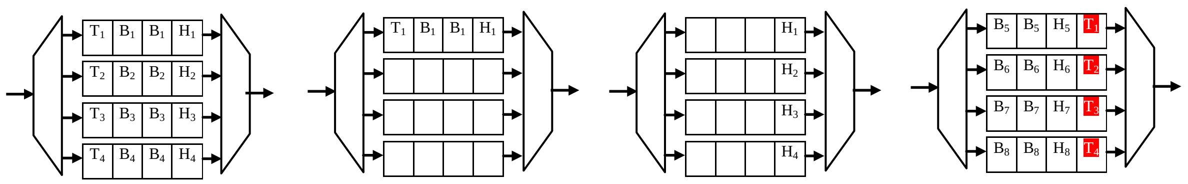
In this report, we have selected to provide buffering organization in the input-ports of channels. The input-ports employ two types of data flow mechanisms commonly known as static and dynamic to organize VCs [4]. In the static mechanism, the buffer slots are statically allocated to the incoming packets, and in the case of dynamic mechanism such as Dynamically Allocated Multi Queues (DAMQ), the buffer slots are dynamically allocated to the incoming packet flits. Most of dynamic VC organizations are table based [13, 14], where a central buffer includes multiple VC queues, and a table keeps the flits of each queue in FCFS order. Basically, the table keeps the address of incoming flits in a FCFS orders.

1.5.1 Problems in Static VC Organization

In the case of static buffering, the numbers of VCs and their buffers remain constant during communication. Various studies have shown that for a large number of static VCs, communication load is difficult to balance across them [1]. Some VCs remain idle while the others are overloaded. Therefore, it is better to allocate more buffer storage to busy VCs and less to the idle VCs. Moreover, static VC buffers are expensive components of routers and they become more expensive for larger flit size or when the VC buffer depth becomes larger. The above drawbacks of static VCs has resulted an adaptive VC organization to achieve VC flow control with maximum buffer utilization.

Now we discuss the conventional VC flow control in a router and a problem associated with buffer utilization. First of all, we define the term, buffer utilization, which is the rate of arrival/departure of flits in a slot buffer per clock cycle. For example, when a slot is empty, or it is full but with the same data per clock cycle, the buffer utilization rate is 0%. Figure 1.7b shows the architecture of a conventional static VC-based router [13]. The input-port stores data in VC buffers in a FIFO fashion as illustrated in Figure 1.7c. The VC identification (*VC-ID*) of a flit is issued before the flit is transferred to the router. The *VC-ID* selects the VC where the incoming flit should be stored. Therefore, at the input of a router, the flit is stored at the tail of the selected VC buffer ("FIFO" sometimes refers to "FIFO buffer" or "buffer of a VC" in this dissertation). When the flit is reached at the head of the FIFO, a request signal is issued to the arbiter. After the arbitration (VC allocation and switch allocation that will be discussed latter), three signals are issued by the arbiter. First of all, *VC-ID* signal is sent out of the router to let the downstream router know about the VC address of the incoming flit. Secondly, the output address of the flit is issued to the crossbar module. Then, a grant signal is issued to the FIFO that leads the flit to reach the crossbar. The flit passes through the crossbar and exit out of the router.

In the VC buffer, the flow of flits follows a wormhole mechanism. Once the flit of a packet occupies the buffer of a VC, no other packet can access it, even when the flit packet is blocked. This type of flow control is problematic in the context of buffer utilization. We illustrate buffer utilization problem through four different scenarios of data flow in a channel in Figure 1.8. In Figure 1.8a, all the VC slots are served and the buffer utilization is maximum i.e. 100%. In Figure 1.8b, one VC is employed in routing and the other VC's slots are idle and cannot participate in the flow result in lower buffer utilization i.e. 25%. In Figure 1.8c, when one slot of a VC is used by a packet, the other slots are reserved and not used by the new packets and will remain empty, which results in a buffer utilization of 25%. In Figure 1.8d, the tail flits ($T_{1,}$ $T_{2,}$ T_3 and T_4) are blocked in each VC. That will block any new packets such as $P_{8,}$ $P_{7,}$ P_6 and P_5. This kind of blocking is called head-of-line (HoL) blocking [4]. No arrival/departure of flits results in a buffer utilization of 0%. It also leads to higher contention and lower performance. The effect of lower buffer utilization on power and area usage is obvious. The idle buffers are useless and only increase the hardware usage. In fact, one of objectives of our VC organization approaches presented in Chapter 3 and 4 is the adaptivity of channels. In adaptive channels, the idle buffers are dynamically used to create new VCs or to increase the buffer depths of active VCs. More VCs and higher VC buffer depth improve the performance and latency.



(a) All VC are served. Buffer Utilization=100%

(b) One VC is employed in routing. Other VCs are idle. Buffer Utilization=25%

(c) One slot of each VC is used. Other slots are reserved and cannot receive new packet due to wormhole routing. Buffer Utilization=25%

(d) One flit (T_1, T_2, T_3, T_4) is blocked in each VC. New packets (P_8, P_7, P_6, P_5) are blocked due to FIFO service (HOL).

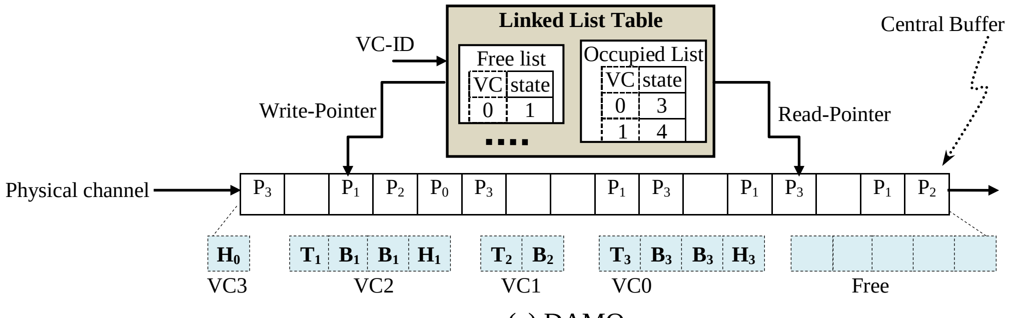
Figure 1.8: Four Different Scenarios in Conventional Wormhole VC Communication.

1.5.2 Head of Line Blocking

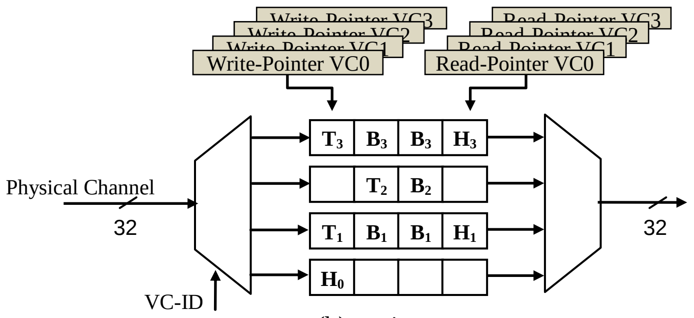
This section introduces the Head of Line (HoL) blocking. In wormhole routing, when a packet passes through a route, the route is reserved and no other packets can utilize that route. This kind of routing cannot avoid traffic congestion when a packet is blocked. In fact, blocking of a packet leads to the blocking of other packets in the channel causing HoL blocking (Figure 1.8d). The HoL blocking causes higher latency and lower throughput. HoL problem can be alleviated by using Virtual Channels [13]. However, the traditional VC approach does not remove HoL problem completely [13, 14, 15].

1.5.3 DAMQ: Dynamically Allocated Multi Queues

The traditional adaptive VC organization and its limitations are introduced in this section. Dynamically Allocated Multi Queues (DAMQ) is a single storage array that maintains multiple FCFS queues. In DAMQ, packet flits are stored in a central buffer consisting of multiple queues. The DAMQ buffers adapt to network traffic by dynamically allocating queue space amongst the output-ports depending on the traffic [4]. The dynamic queues of DAMQ buffers improves buffer utilization of port by sharing its buffer slots among all the VCs of port and allocating more buffer slots to active VCs. Higher VC buffer depth keeps more flits of a packet and leads to a free route of the packet in wormhole NoC communication. The more free routes lead to lower contention and eventually improve the overall NoC performance. Figure 1.9a illustrates a 4-VC DAMQ buffer where the addresses of flits kept in a linked list table. The linked list table records the flit addresses according to their *VC-ID* and in a FCFS orders.



(a) DAMQ



(b) static queues

Figure 1.9: Input-Ports with Dynamic and Static Queues

The micro-architecture of DAMQs can be used for organizing VCs in NoC systems. This technique can also resolve contention, deadlock, or fault tolerance related issues. Despite the performance merits of DAMQ organizations, they have a number of limitations as listed below.

- It has complex hardware due to linked list and dynamic queue management [4, 16].
- Another problem is related to the queue structure that is tailored for deterministic routing. It cannot look after fully adaptive routing since the routing decision for a new packet is made in conjunction with the output queues. With such flow control mechanism, the routing adaptivity cannot be established [4]. Packet flit buffers in NoC routers can be placed at three locations: input-ports, output-ports or both input and output ports [4]. The NoC routers with input-port buffers can easily support adaptive routing as flits resided in an input-port can be processed by following an adaptive methodology. In other words, the incoming flits remain in the input-port buffer until an adaptive routing is implemented (at least one clock cycle) and determines the outputs for the exit of flits. Routers with output-port buffering cannot

implement an adaptive routing mechanism. An incoming flit should be arbitrated as soon as a flit enters the router, and the flits that are buffered in the output-port can only pass through that output (no adaptive routing can implemented). In the initial versions of some DAMQ-based NoCs such as Link-list [14, 17, 18] and ViChaR [13], VC organization techniques had VC buffers at the output-ports. However, some newer versions of Link-list have introduced additional hardware in the form of recruit registers to achieve adaptive routing [4]. Other researchers have also added buffers at the input-ports to support adaptive routing [14].

- Configure limitation is the third problem with some DAMQ mechanisms. For example, three limitations such as limitation in the minimum buffer space of each VC, the number of VC, and the number of flits per packet has been employed in the specifications of some DAMQ schemes [13, 14, 19].
- The Head of Line (HoL) blocking is the forth problem in the communication of some DAMQ schemes. Assuming that a VC (queue) can receive more than a packet, and in case of the packet header blockage, the other packet in the VC has to wait until the blockage removed.
- There are interventions among the VCs of a DAMQ port that can lead to higher traffic congestion as compared to static VCs [20].
- A flit arrival/departure has a large delay due to complex design of DAMQ based VCs.

The initial two problems associated with DAMQ based VCs have also been reported by other researchers and solutions have been proposed [4, 15, 16, 20]. The third and forth problems exist in some DAMQ mechanisms and will be discussed in detail along with our optimal solution in Chapter 4. The last two problems are being introduced in this section. Interventions among VCs exist in all the DAMQ-based mechanisms. As already mentioned, a single storage array maintains multiple VCs of a DAMQ port. Therefore, the communication behavior of a VC directly affects the other VCs. For instance, a blocked VC can occupy the maximum free space of its port buffer, and only a few buffer locations are available to unblocked VCs that lead to higher traffic congestion. We discuss in detail the 5th problem along with our optimal solution in Chapter 4. The longer flit arrival/departure delays mentioned as the last problem exist in table-based DAMQ mechanisms such as Linked list [14, 18] and ViChaR [13] where a central table, containing the registers, is employed for direct data flow. Registers are updated at a clock edge due to which the table-based DAMQ mechanisms take one additional clock cycle than the static VC-based NoC communication [1]. We further discuss this problem through the pipeline stage analyses of two types of port buffers in the following section.

1.5.4 Timing Problem of Adaptive Table-based VC Organization

In this section, we discuss the drawback of table based organizations as compared to static VC organizations. Figure 1.9 shows the architectures of static and dynamic input-ports. The control logic of the static input-port is simpler and each VC can be configured by using a parallel FIFO buffer [20] as illustrated in Figure 1.9b. Each FIFO represents a VC and therefore the number of VCs is equal to the number of FIFOs. The *Read-pointer* and the *Write-pointer* point to the location of FIFO where a flit (data) is read or written respectively. A *pointer* works like a simple counter, which is incremented circularly and continuously for each read and write operation.

The flit arrival/departure is also simpler in static input-port. If arbitration takes one step, the arrival/departure of flits in a squeezed pipelined scheme consumes two clock edges as illustrated in Figure 1.10a. At the entrance of an input-port, an arriving flit is decoded according to its VC identification (*VC-ID*) and by means of de-multiplexer, then it waits to be latched in the FIFO buffer (VC) before the first clock edge. At the first clock edge, the flit is stored in the VC where a request corresponding to that flit is simultaneously issued to the arbiter. At the 2nd clock edge, the arbiter allocates the proper address for the crossbar switch (output) and ID for the downstream router VC then issues a *grant* signal. The *grant* signal causes the flit to exit the router. For proper operation of the decoder at the entrance of the input-port, the *VC-ID* should be issued earlier than the latching of the flit in the buffer. Assuming that the flit and its *VC-ID* are transferred at the same clock transition, each flit arrival/departure takes a two-clock event delay in the static VC router. We have assumed that the FIFOs are dual-port, where the arrival of a flit can coincide with the departure of another flit.

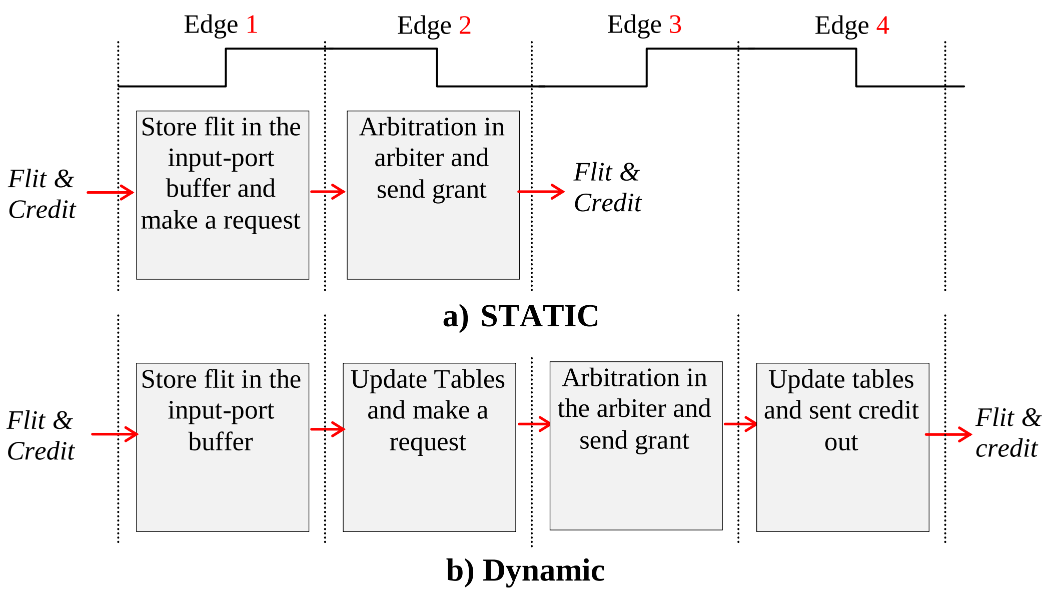

Edge 1
Edge 2
Edge 3
Edge 4
Flit & Credit
Store flit in the input-port buffer and make a request
Arbitration in arbiter and send grant
Flit & Credit
a) STATIC
Flit & Credit
Store flit in the input-port buffer
Update Tables and make a request
Arbitration in the arbiter and send grant
Update tables and sent credit out
Flit & credit
b) Dynamic

Figure 1.10: Staic vs. Dynamic Input-Port Pipelines

In the case of dynamic VC input-ports, the VC buffers are allocated dynamically based on the traffic resulting in more complex control logic. Linked-List based DAMQ has been employed as a conventional DAMQ in many research projects [14, 17, 20, 21]. Using this mechanism, a single buffer (queue) maintains multiple VCs, and the data flow is directed by Linked-List tables as illustrated in Figure 1.9a [14]. The *Read-pointer* and *Write-pointer* are updated based on the contents of the linked list tables. In a squeezed pipelined design, when arbitration takes one step, the arrival/departure of flits will take four clock edges as illustrated in Figure 1.10b. A head flit arrives at the input-port and waits to be latched in the VC (buffer). Then the flit is latched into the input-port buffer at the first clock edge. In the 2nd clock edge, the Linked-List tables are updated according to the *VC-ID*, which leads to a request signal being issued to the arbiter. In the 3rd clock edge, the arbiter assigns a proper address for the crossbar switch (output) and ID for the VC before issuing a *grant* signal. The *grant* signal causes the flit to exit the router, as well as the linked list tables are updated at the 4rd clock edge. In a Linked-List DAMQ based VC organization, the read and write pointers cannot be updated at read or write events. Instead, they will be updated one clock event after the read and write (i.e. after updating the tables). However, in the static VC queue, the read and write pointers can be incremented at the read or write events. This causes the pipeline stages in DAMQ table-based input-ports to be one clock cycle longer than those of static input-ports. The same communication characteristics are expected for other table-based DAMQ mechanisms e.g. ViChaR [13]. The ViChaR has a central table that contains registers to direct the data flow. Registers are updated at one clock edge that results in the usage of one additional clock cycle. However, the VC organization approach presented in Chapter 4 does not employ tables. Moreover, its flit arrival/departure delay is equal to that of static VCs but with all the advantages of dynamic VCs.

1.6 Data Flow Arbitration

In this section, we introduce the data flow arbitration that is the process after buffering. It also introduces a state of art arbiter in Chapter 5. After buffering a flit in a VC of an input-port, VC issues a request to the arbiter for accessing shared resources. The structure of arbiter becomes more complex when an NoC utilizes VC mechanism in its data path (extra hardware for VC and switch allocators). The arbiter can perform arbitration and allocation in four pipelined stages as follows. First, the route must be computed to determine the output-port (or ports) to which the packet can be forwarded. Then, a downstream router VC (VC in router's input-port) should be allocated. When the flit's buffer space is booked in the downstream router, the flit can begin to compete for access to the crossbar switch. Once a route has been determined and a downstream router VC allocated and the crossbar switch configured, the flit is forwarded over this VC to the downstream router on the route. For explaining, consider the case illustrated in Figure 1.11 where the HF flit (header flit of a packet) is in VC1 of input-port 5 of router 3.

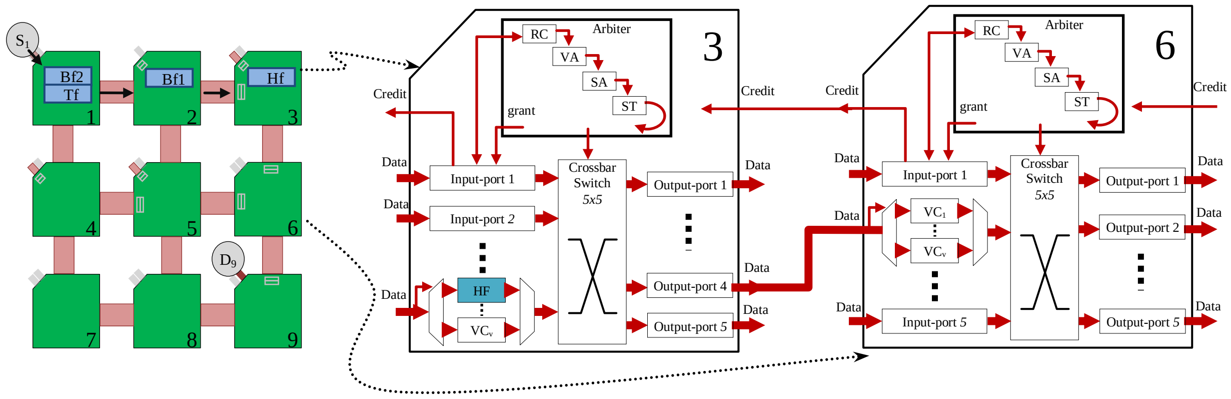


Figure 1.11: HF Flit in VC1 of Input-port 5 of Router 3 Traveling to Input-port 2 of Router 6.

To advance the flit, HF to router 6, a space in a buffer in the input-port 1 of router 6 must be allocated, and HF must win the allocation to traverse the crossbar switch. To begin advancing the HF, the route computation is first performed to determine the output-port to which HF can be forwarded. Based on XY routing (see Section 1.2) the output-port 4 is assigned. Then, HF requests a VC from the VC allocator (assume VC2 of input-port 1 of router 6). When the buffer space of HF is reserved in router 6 along the output to the downstream router, the flit can compete to access the crossbar switch by means of switch allocator. Once the output-port has been determined, a VC is allocated, and the crossbar switch is configured, the HF flit can travel from output-port 4 to the VC2 of input-port 1 of router 6. We explain the above stages and their timing processes in the following section.

1.6.1 Arbiter Pipeline Stages

The pipelined stages of a typical VC-based arbiter are illustrated in the timing diagram of Figure 1.12. Each flit of a packet must go through the stages of Routing Computation (RC), Virtual channel Allocation (VA), Switch Allocation (SA), and Switch Traversal (ST) [2]. The RC and VA stages perform computation only for the header flit (once per packet). Body and tail flits pass through these stages without RC and VA computation. In fact, the SA and ST stages operate on every flit of a packet, and only the header flit passes through all the stages.

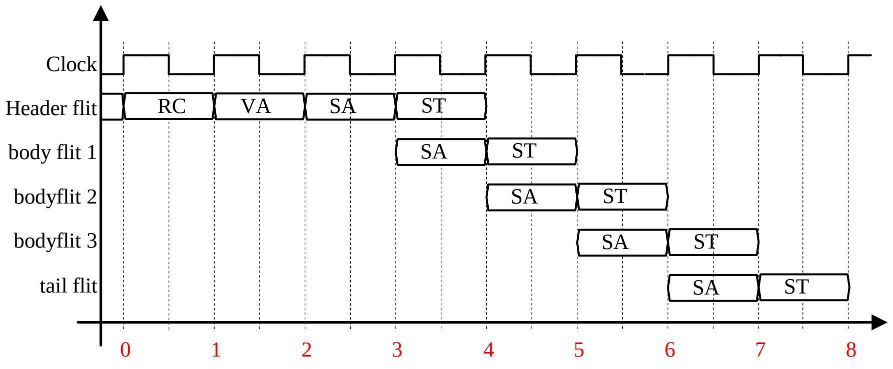


Figure 1.12: The pipelined stages of a typical VC-based arbiter. A packet consists of 5 flits takes 8 cycles to be arbitrated in case of no stall in the communication.

In our design, the first three stages i.e. RC, VA and SA proceed in parallel and in one clock cycle as illustrated in Figure 1.13. However, in other designs, each stage may take one or more than a cycle. As one can notice in Figures 1.12 and 1.13, ST stage is the last stage and it needs a separated clock cycle. It can work concurrently with the first stage of the following flit [22]. We will discuss the advantage of our pipeline mechanism later. The arbitration process begins when the header flit of a packet leads a request to be issued to the arbiter, assume at cycle event 0 of Figure 1.12. During the following four cycles, the request of flit remains activated that leads all the four stages to proceed. The header flit information is used by the RC stage to select the requested output-port at the clock event 1. The result of RC stage along with the states of downstream router VCs are used as inputs of VA stage to pick a free downstream router VC for the packet at the clock event 2.

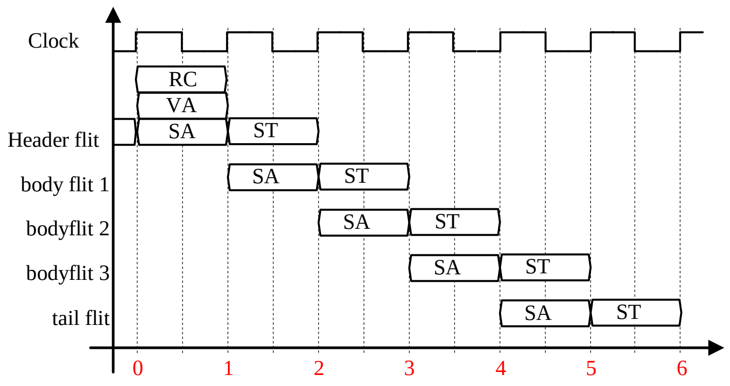


Figure 1.13: The pipelined stages of a typical virtual channel arbiter. A packet consists of 5 flits takes 6 cycles to be arbitrated in case of no stall in the communication.

The information provided by RC stage along with the information of requested downstream router VC are used by SA stage to determine the winner inputs VCs at the clock event 3. At the clock event 3, two tasks such as the issuance of *grant* signal and crossbar address are performed to prepare the route, where the flit passes at the following clock cycle. Therefore, the request of next flit after the header flit can be processed during the fourth cycle. The flit travels to a downstream destination by means of proper handshaking signals during the 4th cycle (we assume a credit-based flow control). At the end of RC and VA stages, the information generated by RC and VA stages are saved in a register table and will be used for the following (body and tail) flits of the packet. For example, the result of VA stage is recorded to be issued as a new VC-ID for packet flits. When a body or tail flit makes the request, the recorded RC and VA information are used by the SA stage, and these two stages are bypassed as illustrated in Figure 1.12. From this point until the release of the channel by the tail flit, the recorded RC and VA information remains unchanged. Therefore, the four stages proceed through two distinct frequencies: packet rate and flit rate. RC and VA stages are performed once per packet. On the other hand, SA and ST stages are performed per flit basis.

1.7 Motivation

The motivations for the research conducted and presented in this thesis are listed as follows.

- Network-on-Chip architectures are viewed as a possible solution to meet the wiring challenges of MPSoC systems
- NoC design that consumes minimal power, IC area but with higher performance is a necessity for SoC design especially for low power high performance applications.
- Current NoC router and NoC system designs are not optimal.
- The main problem with the current NoC design is related to lower performance under high contention due to traffic congestion.
- Router is the key component of NoC, and if one improves its design, it will improve the overall NoC performance.
- Both NoC performance and energy budget depend heavily on the routers' buffer resources. One must use the buffer cleverly and intelligently for NoCs e.g. employ Adaptive VCs.
- Adaptive VCs also have maximum buffer utilization of the router.
- Current DAMQ buffer design suffers a number of problems such as complexity, lower buffer utilization, setup limitation, and HoL blocking.
- Arbitration is the other important activity in NoC routers. It can have some problems such as complexity, lower speed, weak fairness, traffic starvation, and pipelined difficulty.
- The above drawbacks and points related to current NoCs have motivated us to investigate the high performance components of NoC router including input-port and arbiter.

1.8 Objectives

NoC architectures have been commonly presented in Globally Asynchronous Locally Asynchronous (GALS) design style [23], and in this thesis we have also followed the GALS style for our NoC router design. In NoC GALS architectures, the routers are locally synchronous, but the NoC architectures are globally asynchronous, i.e. there can be different clock rates for routers. In other words, the routers are independent in terms of clock design, and the faster clock rates of routers leads to faster NoC. The main objectives of the research presented in this dissertation are to design, present and evaluate an efficient NoC wormhole router. The packet-based wormhole routing has been introduced as a viable communication mechanism being employed in NoCs. The conventional wormhole routing flow is prone to contentions and in some cases deadlocks. One of the traditional ways to alleviate contention and to remove deadlock is to the introduction of VC organization. A traditional form of VC organizations has been of static type. The numbers of VCs stay constant during communication for static VCs. The static VC organization is also expensive in terms of higher number of buffer cells and suffers from lower buffer utilization. To solve the problem related to static VC organization, a commonly used dynamic VC organization namely DAMQ has been introduced. Despite the performance merits of DAMQ organizations, they have a number of limitations such as complexity, lower buffer utilization, lower frequency, longer pipelined stage, configure limitation and HoL problem. The limitations of static and dynamic VC organizations have encouraged us to present two efficient adaptive and dynamic VC organizations techniques.

The arbitration is another important module requiring new and novel architecture for efficient NoC router design. The arbitration organization is implemented in the arbiter module of router that can have four pipelined stages: Rout Computation (RC), Virtual channel Allocation (VA), Switch Allocation (SA), and Switch Traversal (ST). The arbitration and allocation functions are performed inside the VC and switch allocators of arbiter module. The conventional NoC arbitrations suffer from some drawbacks such as complexity, lower speed, weak fairness, traffic starvation, and pipelining problems. The limitations of conventional arbiter organizations have encouraged us to investigate design and present an efficient and fast arbiter.

Finally, by employing novel VC buffering and arbitration organizations, our main objective is to improve the performance and hardware metrics of routers and NoC systems. The experimental results support the theoretical concepts of our proposed VC organization and arbitration approaches for efficient NoC system.

1.9 Thesis Organization

- Chapter 2 reviews some important past research works related to DAMQ based VC organizations and round-robin arbiter architectures.
- Chapter 3 presents and evaluates our adaptive and efficient VC organization based on Statically Adaptive Multi FIFO (SAMF). The SAMF VC buffers are static during a specific time of communication but subsequently adapted to the traffic demand.

- Chapter 4 describes our Efficient Dynamic Virtual Channel (EDVC) organization and its novel features. The EDVC mechanism utilizes the common features of DAMQ input-port to create a dynamic flow control.
- Chapter 5 provides a detailed presentation and evaluation of our novel and efficient router architecture. The router utilizes two new components including an RDQ input-port and IRR (Index-Based Round Robin) arbiter.
- Chapter 6 will discuss the thesis conclusions and our future works.

Chapter 2

Previous Research Work

Three types of NoC research is reviewed and investigated in this chapter. First we discuss some approaches related to adaptive buffer organization. Specifically, two major components associated with the conventional buffer organization i.e. FIFO and DAMQ buffers are described in detail in Sections 2.2 and 2.3 in an effort to highlight their problems. Our implementation of a well-known DAMQ architecture i.e. Link-List is presented in detail in Section 2.4. Some researches that focus on the design and organization of NoC routers are investigated in Section 2.6. Then various micro-architectures of a Round Robin arbiter used in the NoC router is presented and discussed in Section 2.7.

2.1 Buffer Organization

FIFO or Queue is frequently utilized for buffer organization in NoC routers. They temporarily stores messages in the form of first come first serve (FCFS) order until network resources become available. Commonly used terms, “queue” and “FIFO” sometimes have the same meaning when the concept of first in first out is considered. However, in terms of architecture, first-in-first-out queue is mostly referred to serial or parallel FIFOs, and the queue is referred to all the buffers with FCFS concept that comprises FIFO buffers too.

In industrial and academic research, many queue architectures have been proposed and the FIFO, Circular Queue (CQ), dynamically allocated multi-queue (DAMQ) and their variants are well-known queue designs [4, 15, 17, 18, 24, 25]. In the following section, the FIFO and CQ organizations are discussed in detail under the name of serial and parallel FIFO [4].

2.2 Serial and Parallel FIFO Architecture

There are two types of FIFO designs and architectures: serial and parallel [4, 7, 13, 26, 27, 28, 29]. The serial FIFO (such as shift register) that works by fall-through principle has been the first FIFO generation as shown in Figure 2.1. However, the architecture of conventional FIFOs is constantly being improved. Currently, most of the FIFOs are parallel, which is an appropriate mechanism to increase the number of stored words along with faster speed [7]. This trend is suitable for network on chip due to two main reasons. The first reason is related to the fall-through concept where the newly arrived data unit is store at the tail (cell) of the FIFO, and at each shift request it is shifted one step toward the head of the FIFO queue. In this way, the data units are shifted through all the storage location at each request. This concept has three drawbacks of long fall-through delay, bubble cells and high dynamic power consumption. The first drawback is due the fact that when the FIFO's capacity is increased, its fall-through time will increase leading to higher FIFI latency [7]. In fact, the minimum latency of a FIFO depends on the depth of physical FIFO rather than the number of stored items. The second drawback of bubble cells in the FIFO is illustrated in Figure 2.1. The bubble cells can occur when the data input/output rates are different. The third drawback is the dynamic power consumption due to data shifts from tail to the head of FIFO. Serial FIFO is simpler, but it is unsuitable for on-chip implementation [26]. The FIFO architecture should not shift the data items through all the memory locations. In other words, the arrival packet should be stored at the front of empty cell rather than at the tail of a queue.

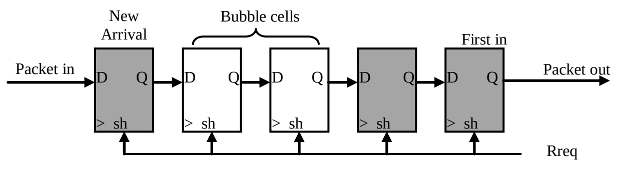


Figure 2.1: Conventional Shift Register (serial) FIFO.

To solve these drawbacks of deep FIFOs, a parallel FIFO mechanism that relies on read and write pointers has been introduced [13]. We have already discussed this mechanism and the architecture of parallel register-based FIFO in Chapter 1, which is illustrated in Figure 1.6. The same style is also used in SRAM-based FIFOs as shown in Figure 2.2. The write and read address ports of the SRAM are the selection ports of de-multiplexer and multiplexer while the *Read-pointer* and *Write-pointer* work as counters.

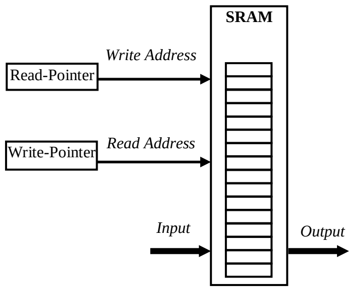


Figure 2.2: SRAM-Based FIFO Using Parallel Style.

The register-based buffers are usually more expensive in terms of power because they use more transistors than SRAM-based buffers [27]. However, register-based operations (read, write, shift) only involve the occupied cells, while SRAM operations (read, write) involve all the cells due to global bitline and wordline wiring. Therefore, register-based buffer may consume less energy than SRAM based buffers when the buffer utilization is below a certain threshold and higher energy when the buffer utilization is above the threshold [27]. Figure 2.3 shows the threshold utilization of different register-based buffers with different sizes and technologies.

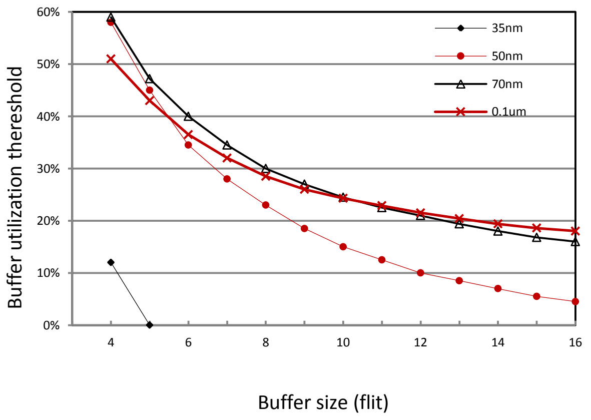


Figure 2.3: Buffer Threshold Utilization vs. Buffer Size.

The register-based implementation is still viable at 0.1μm technology with relatively smaller buffer size and lower buffer utilization, but it is not a good choice for 35nm technology. This is mainly due to increasing static power for 35nm or lower technologies, where the advantage of fewer activities is completely diminished by the disadvantage of more transistors [27]. As the buffer capacity increases to dozens of flits, the register-based implementation becomes inefficient due to larger chip area occupied by the buffer [26]. The schematics of a dual-port SRAM cell and a D-type flip-flop that are used for large-capacity FIFOs are shown in Figures 2.4 and 2.5 respectively. Assuming that a NOT gate and a NAND gate need two and four transistors in their structures respectively [30]. A SRAM cell occupies only a third of the D-type flip-flop register area. Now-a-days, NoC buffers are mainly implemented by SRAM due to the area and power cost and the availability of the corresponding Intellectual Property (IP) cores [24]. The above facts have encouraged us to use SRAM-based buffer and parallel style mechanism in all the proposed designs presented in this dissertation.

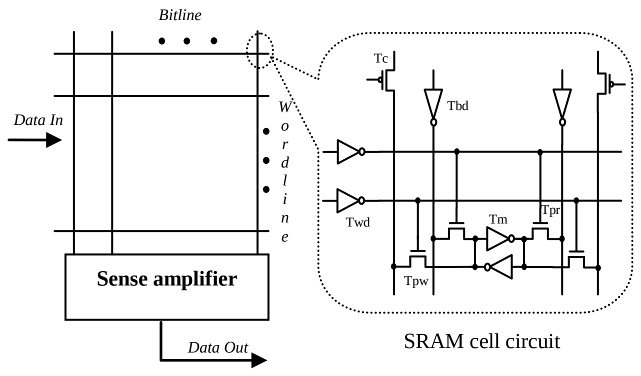


Figure 2.4: SRAM-based FIFO.

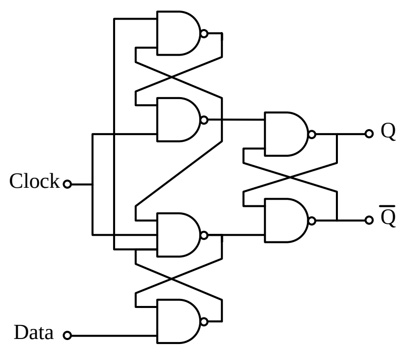


Figure 2.5: A Positive-Edge-Triggered D-FF.

2.3 DAMQ Buffer Organization Research Work

DAMQ that is a unified and dynamically-allocated buffer structure was originally presented by Frazier and Tamir [18]. It is a single storage array that maintains multiple FIFO queues. The DAMQ mechanism dynamically allocates multiple queues on a physical channel. In other words, the DAMQ buffers are able to efficiently adapt to network traffic by dynamically allocating queue space among the output-ports according to the network traffic [4]. These dynamic queues of DAMQ buffers lead to maximize buffer utilization. The DAMQ organization can be used in the VC organization of NoC. This technique is able to solve or alleviate the other NoC issues such as contention, deadlock, HoL blocking or fault tolerance. Jamali and Khademzadeh [15] has used DAMQ buffer scheme for fault tolerance in NoC systems. There are some drawbacks related to initial DAMQ scheme presented by Frazier and Tamir [18]. We discuss these drawbacks in the following section. The first drawback of scheme is that the packets are stored into the queues of a multi-queue of the output channel for routing. Therefore, in the case of blockage of the output channel, the packets destined to that output-port become blocked. According to Choi and Pinkston, this type of blocking is not HOL [4], but we argue that it is HOL blocking as explained earlier in Section 1.4.5. In fact, packets in an output channel have different destinations, and they travel to different output channels in the downstream router. Therefore, if the head of line of these packets is blocked due to the blocking of its route in the next downstream router, the remaining packets will be blocked even though their routes are open in the next downstream router. Figure 2.6 illustrates a HOL blocking case where eastward output channel of router1 is blocked due to the blocking of P1. The packets P2 and P3 are blocked despite the fact that their routes are open in the downstream router.

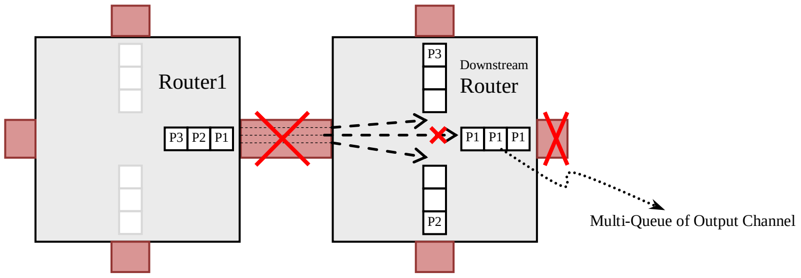


Figure 2.6: Head of Line Blocking in DAMQ Output Channel.

The second problem of scheme is again related to the location of queues in output channel. In fact, the queue structure is tailored more for deterministic routing algorithms than for fully adaptive routing algorithms. In the scheme, a routing decision for a new packet is made in order to assign the packet to one of the output queues. This forces the packet to be routed only through that output. In such a flow control, the routing adaptivity cannot be established [4]. The third problem of initial DAMQ approach is that there is no reserved space dedicated for each output channel [18]. The packets destined to one specific output-port may occupy the whole buffer space. Therefore, the new packets destined to this output-port have no chance to get into the buffer [15]. The fourth problem is related to its hardware complexity caused by the linked-lists and dynamic queue management utilized in the scheme [4]. Despite the performance merits of Link-List mechanism utilized in this scheme, it suffers from a few complications and limitations that we will discuss in detail in Section 2.4. In fact, one of the objectives of our approach presented in Chapter 4 is to implement a DAMQ buffer without Link-List mechanism.

Different buffer architectures are proposed to overcome various limitations of DAMQ used in NoCs. Dynamically allocated multi-queue with recruit registers (DAMQWR) and virtual channel dynamically allocated multi-queue (VCDAMQ) are proposed in an effort to overcome some of the drawbacks of DAMQ [4]. DAMQWR uses DAMQ architecture with some recruit registers to implement adaptive routing for on-chip communication. The main recruit registers assign the packets of blocked sub queues to less congested sub queues. However, in addition to hardware overhead, DAMQWR method has additional delays due to recruit register updates and packet recruit operations. The queue organization that resembles DAMQ in the VCDAMQ method can efficiently adapt to an unbalanced traffic load amongst the VCs by dynamically allocating queue space to virtual channels. In fact, the difference between the VCDAMQ and the traditional DAMQ lies in the fact that the sub queues of VCDAMQ are associated with router VCs while those of the DAMQ are associated with the router output-ports.

Lai *et al.* introduce a Link-List based DAMQ architecture with congestion awareness [20]. They have added congestion-avoidance logic to the arbiter for predicting congestion situation at the immediate neighbors. Lai *et al.* also proposed DAMQ architecture to remove HoL blocking [21]. Once a packet faces an HoL blocking, an extra VC is allocated. This method to remove HoL blocking and congestion is expensive and complex when compared with our approach presented in Chapter 4. Our approach intrinsically avoids congestion that also takes care of HoL blocking. Zhang *et al.* presented a novel multi-VC dynamically shared buffer (DAMQ-PF) for NoC systems [24]. Their design has a small pre-fetch buffer, which is used for each VC. This buffer can store the data read from the shared buffer to provide dedicated storage for each output-port. This allows continuous and concurrent access of the shared buffer without any delay. Zhang *et al.* have also proposed a fair credit management method to avoid the situation where a single VC can occupy the shared buffer exclusively [24]. Liu and Frias proposed a new DAMQ buffer organization scheme with a reserved space for each of the virtual channels [31]. The main feature of scheme is that there is a reserved space dedicated for each virtual channel. As shown in Figure 2.7, two buffer slots are reserved for each virtual channel before the buffer accepts any incoming flit.

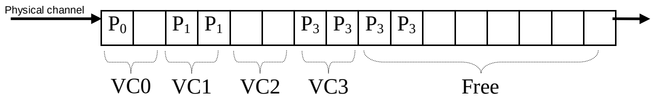


Figure 2.7: Reserved Space for Virtual Channels [19].

Different schemes have been used to implement DAMQ mechanism [13, 14, 16, 18, 24]. All of these schemes are either expensive in terms of hardware or inefficient due to data dependency (specifically when the packet becomes bigger). We describe some of important DAMQ mechanisms in the following sections.

2.3.1 Virtual Channel Regulator (ViChaR)

Nicopoulos *et al.* introduced a centralized shared buffer architecture called the Virtual Channel Regulator (ViChaR) [13]. This design avoids using the linked list mechanism but its control logic is expensive. In spite of the advantage of supporting a large number of adaptive VCs (as big as the number of slots in a channel buffer), theoretically ViChaR cannot assign a specific room to each VC. In some cases, this will create a deadlock or high traffic contention. ViChaR dynamically allocates VCs and grants new flit on a First-Come-First- Served (FCFS) basis, and there is no limitation for each VC. Therefore, in the case of blocking, a packet can occupy the entire slots of a channel buffer and prevents any new packet to pass through the router. If the blocking of that packet continues, all the upstream routers will be occupied by the packet, and no other new packet can pass through the route. This blocking can be spread through the NoC system and create deadlock. Technically, this problem is due to the specification of ViChaR structure where the VC size varies from one to maximum size of buffer of a channel. Another drawback of the approach is a huge NoC hardware in some configurations. In ViChaR method, the information of incoming buffer is saved in a table and two trackers as illustrated in Figure 2.8.

Slot availability tracker

Slot	Available
0	0
1	1
2	1
3	1
4	1
….	…
15	1

(a) 1 denotes the slot in input port memory is occupied, and zero means it is empty. 16 1-bit registers

VC Control table

VC ID	Direction	Header	Data	Data	Tail
VC0	East	1	3	N	N
VC 1	South	2	4	N	N
VC 2	West	N	9	10	N
VC 3	East	N	N	N	15
VC 4	N	N	N	N	N
….	….	….	….	….	….
VC 15	N	N	N	N	N

(b) N denotes slot is free assume input-port memory has 16 slots. When a flit arrives its information recorded in the table. 16 19-bit registers

VC availability tracker

VC ID	Available
0	0
1	1
2	1
3	1
4	1
….	…
15	1

(c) 1 denotes the related VC in input channel has flit, and zero means it is empty. 16 1-bit registers

Figure 2.8: One big table and two trackers used in ViChaR method [9].

The VC control table module that holds the slot IDs of all the current flits becomes very large when the flit size is small or the packet size is big. Another drawback of approach is slower speed of system operation. As mentioned before, ViChaR can support a maximum number of VCs as the number of buffer slots (BS) in the channel buffer. This requires the arbiter in both VC allocation and switch allocation stages to match the BS size. Such a size of the BS may create a latency bottleneck in the critical path of arbiter and consequently the router, which can limit the NoC frequency [16]. We will evaluate and compare it with our approach in Chapter 4.

Several features of ViChaR architecture have encouraged some researchers to employ this in their designs. Nicopoulos *et al.* has presented the design of an intelligent buffer that logically reorders the entries in a FIFO buffer to minimize overall leakage power consumption [32]. In this design the buffer slots are first classified based on their leakage characteristics. Then, the write module attempts to direct incoming flits to the least leaky slot. Moreover, all unused slots are supply-gated using sleep transistors to minimize leakage power consumption. Xu *et al.* presented another application that employs ViChaR architecture [16] where VCs are assigned based on the designated output-port of a packet in an effort to reduce the HoL blocking. Unlike ViChaR, this design uses a small number of VCs. In other words, their buffer design is similar to ViChaR except that each VC can store multiple packets and the number of VCs is fixed. Their buffer design uses a smaller arbiter and VC allocation scheme is hybrid type i.e. in-between static and dynamic.

2.3.2 Self-Compacting Buffers

An approach called "self-compacting buffers" is presented by Park *et al.* to implement DAMQ switching elements [25]. There is no reserved space dedicated for any VC, that is, the first drawback of approach. In fact, a VC can receive as many as flits to occupy the whole of channel buffer. We will discuss in Section 4.5.3 that this specification of mechanism leads to deadlock. Data in self-compacting buffer is stored in a FIFO manner within the region for each VC. When an insertion of a flit requires space in the middle of the buffer, the required space will be created by moving down all the flits which reside below the insertion address. Similarly, when a reading operation conducted from the top of a region, data removed from the buffer may result in empty space in the middle of the buffer, then the data below the read address is shifted up to fill the empty space. For example, assume a scenario in self-compacting buffer as illustrated in Figure 2.9. Assume the order of data in each VC as well as in the buffer is from left to right. Figure 2.9a illustrate a condition that three packets A, B and C have been stored in VC0, VC1 and VC2 respectively. Assume VC1 receives a new flit, B2 as illustrated in Figure 2.9b. B2 should be stored in right side of B1, as already mentioned that the data is stored in a FIFO manner self-compacting buffer. For this reason, the data in right side of B1 should be shifted a slot to the right. Figure 2.9c illustrates a read from the buffer. The flit, A0 is read, so all the data in the buffer should be shifted a slot to the left.

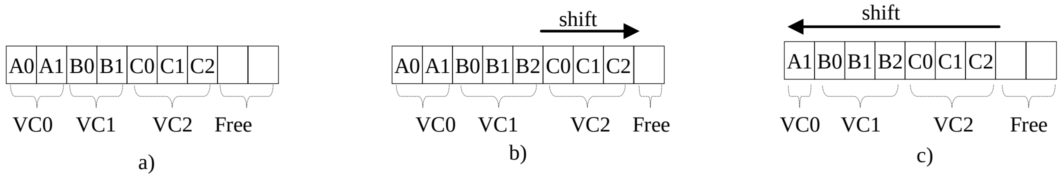


Figure 2.9: Write and Read Scenario in Self-Compacting Buffer

The authors claim that their approach is efficient because the amount of hardware required to manage the buffers is relatively small when it offers high performance. However, this approach has some drawbacks. In terms of behaviour, the self-compacting mechanism looks like the fall-through methodology already discussed in Section 2.2. Therefore, it can have some of drawbacks of fall-through buffer e.g. long delay and high dynamic power consumption. The first drawback is due the fact that when the buffer capacity is increased, its fall-through time will increase, leading to longer latency of the buffer [7]. In fact, the latency of self-compacting buffer depends on its depth rather than the number of stored items. The second drawback is high dynamic power consumption due to data shifts in the buffer. To solve the above drawbacks of self-compacting buffer, Frias and Diaz proposed a novel buffer [33]. They proposed a new cell that has the capability of performing all the required data moves. The novelty of their approach is at the transistor level rather than the gate level. In other words, to evaluate this approach, one has to have their specific IP cell.

2.3.3 Mask-based

Mask-based approach has been introduced to implement DAMQ by Evripidou *et al.* [14]. It is cheaper hardware-wise but slower in terms of performance. In fact, the credit for each VC follows a round robin scheme and it is synchronized with the clock to send out its packet flit. Therefore, Mask-based approach leads to synchronous communication that is not much useful for NoC routers. Evripidou *et al.* have also presented a new version of Link-List mechanism, which mimics the DAMQ organization presented by Frazier and Tamir. As compared to Mask-based buffer, the Link-List buffer is expensive in terms of hardware but leads to higher communication performance [18]. In the following section, we describe the architectural detail of a Link-List based organization that follows the protocol of Evripidou approach.

2.4 Link-List based DAMQ Organization

In this section, we present our implementation of the Link-List based DAMQ (LLD) mechanism. The micro-architecture of LLD input-port of an NoC router is presented here to illustrate the complexity and cost of LLD mechanism as compared to our proposed VC architectures presented in Chapters 3 and 4.

2.4.1 LLD and Static Read and Write Mechanism

We compare the read and write mechanism in static and dynamic queue types before presenting the LLD architecture in more detail. Please note a queue refers to a VC in this dissertation. In Figure 2.10, the VC implementation of a physical channel is illustrated through two queue types: static and DAMQ (dynamic). In a static queue, the buffer slots are statically allocated to incoming packets where in the case of a DAMQ queue, the buffer slots are dynamically allocated to incoming packets. As mentioned earlier, the pointers of each queue in Figure 2.10a are updated circularly and sequentially for each read and write to the queue. However, DAMQ technique updates the contents of read and write pointers by means of a linked list format of saved flit addresses. The linked list information determines the order of VCs in the channel buffer as well as the order of slots in each VC [18]. For each channel, a table keeps the linked list addresses of the queues buffer.

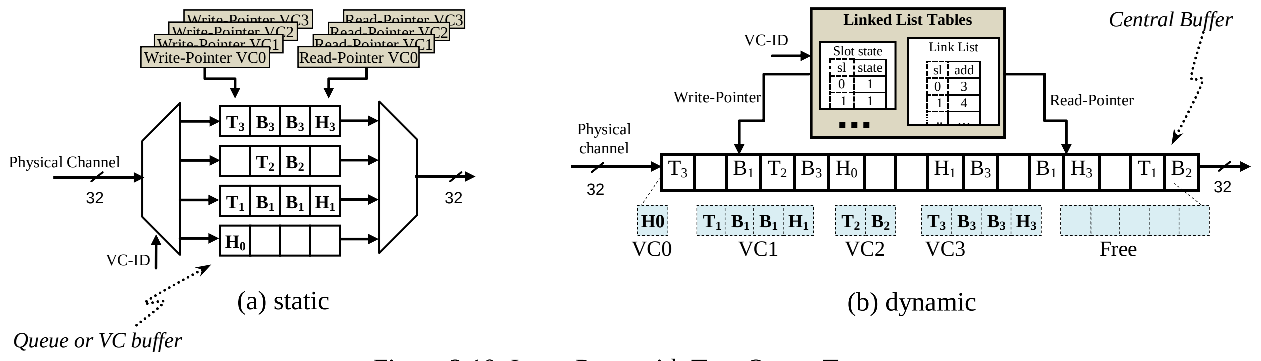


Figure 2.10: Input-Ports with Two Queue Type

The read pointer is updated according to the information stored in linked list table and points to the first slot in a queue. For all the queues, there is one slot state table that is used to keep track of free slots available for incoming packets. The write pointer is updated according to the slot state information and point to an unoccupied slot where incoming flit can be stored. For each read and write of the buffer, the linked list and the slot state tables are updated.

2.4.2 LLD Router

An LLD router consists of input-port modules, an arbiter, and a crossbar switch as shown in Figure 1.1. The LLD router input-port consist of a central buffer, five lookup tables, and some other logic circuits and ports as illustrated in Figure 2.11.

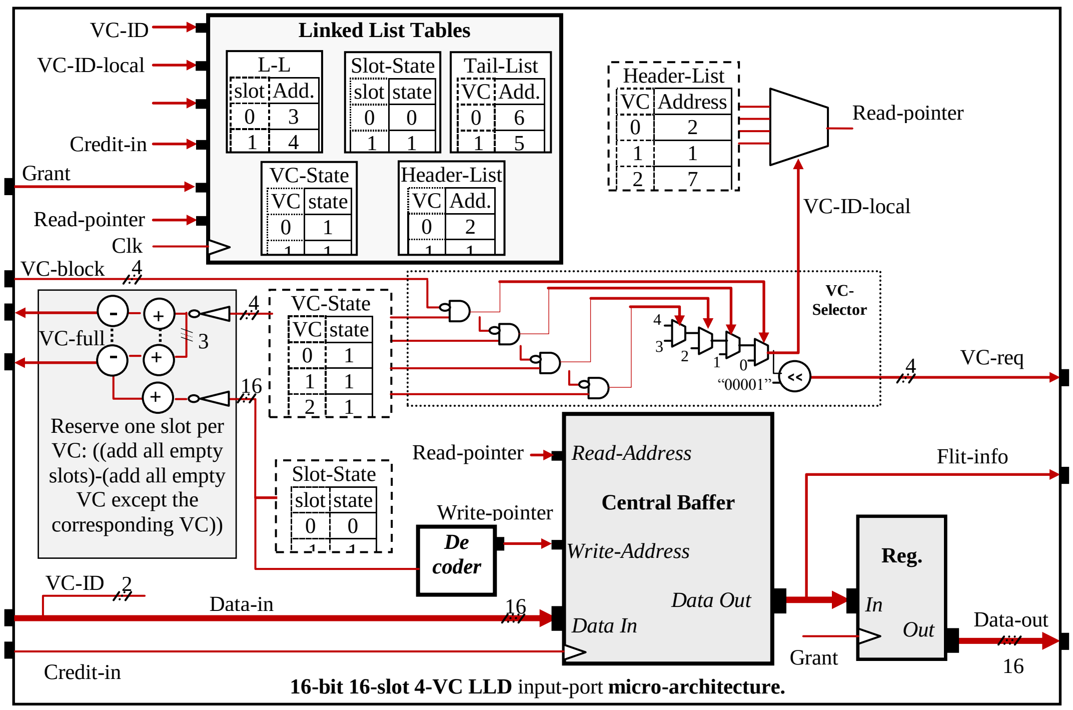


Figure 2.11: LLD Based Input-Port.

A slot of the central buffer comprises of a packet flit, where the slot size of the buffer is equal to the flit size. On the activation of *credit-in*, the data is stored in the slot pointed by the *write-pointer*. The data pointed by the *read-pointer* appears at the central buffer output. Five lookup tables are used to implement the LLD router where three of these tables are shown in Figure 2.12. The *VC-State* and *Slot-State* tables keep a Boolean value for each VC and Slot (empty/occupied). The *Header-List* table keeps the addresses of slots that contain the header flits of VCs. The *Tail-List* table keeps the addresses of slots that point to the tail flits of VCs. The L-L table keeps the addresses of the next slot of each slot, or it links the addresses of slots that are associated with each VC in a FIFO manner. The *Slot-State* table has a record of the occupied slots in the central buffer.

L-L

Slot	Linked addresses
0	3
1	4
2	7
3	6
⋮	⋮
15	N

(a) Link addresses of VCs
16 registers (4-bit)

Header-List

VC	address
0	2
1	1
2	7
3	-

(b) Addresses of first flit of each VC
Four registers (4-bit)

Tail-List

VC	address
0	6
1	5
2	7
3	-

(c) The address of last-stored flit of each VC

Figure 2.12: LLD Router (4-VC and 16-slot) – Lookup tables

2.4.3 LLD Communication

We are employing an asynchronous communication among routers, destination and source cores. A credit based handshaking is used to establish communication between the source, intermediate and destination routers. A credit signal is generated when a source core sends a packet flit. In the case of a destination router, the credit signal causes the data to be stored in the input-port buffer. If the buffer is full then an acknowledge signal, *VC-full* is sent back to the source, signaling it to cease sending flits to the input-port. Following steps describe the communication of flits in the router's input-port, where Figure 2.13 illustrates the timing diagram of the communication.

1) Data and VC identification (*VC-ID*) appears at the input-port of the router at clock event 1.
2) At clock event 3, *Credit-in* signal becomes high and leads the storage of data in the input-port buffer.

3) At clock event 4, all the tables of the input-port are updated according to *VC-ID* and the *request* signal is set, which causes the arbiter to read the flit-data information.
4) At the positive clock edge 5, a *grant* signal is issued after arbitration. This leads the data and its new *VC-ID* to exit the input-port.
5) All the tables are updated and *request* signal is reset at clock event 6.
6) A high level of *grant* signal causes the *credit-out* to be set and the *grant* to be reset at the positive clock edge 7.
7) A high level of *credit-out* signal will also reset the *credit-out* at the positive clock edge 9.

The pipelined communication illustrated in Figure 2.13 shows that in the case of LLD routers, flit-data is stored for two clock events and transferred at two clock events. The tables are updated at the negative clock edge, and the signals are detected and issued at the positive clock edge.

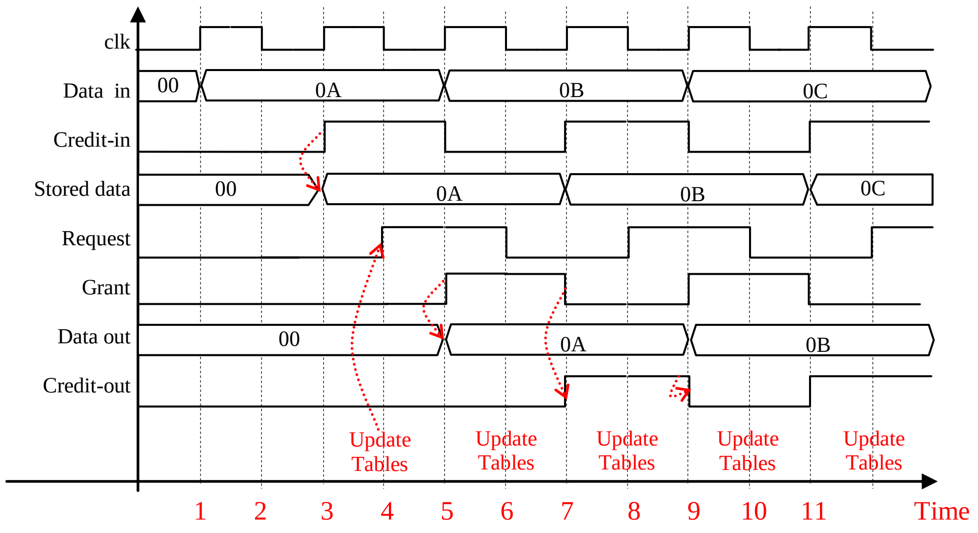
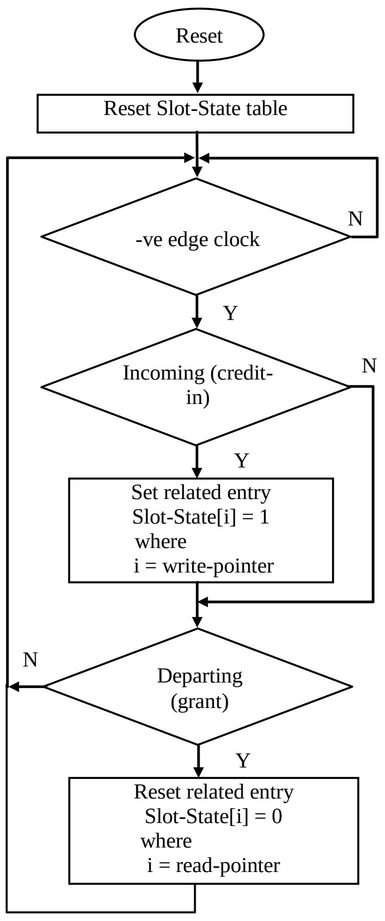


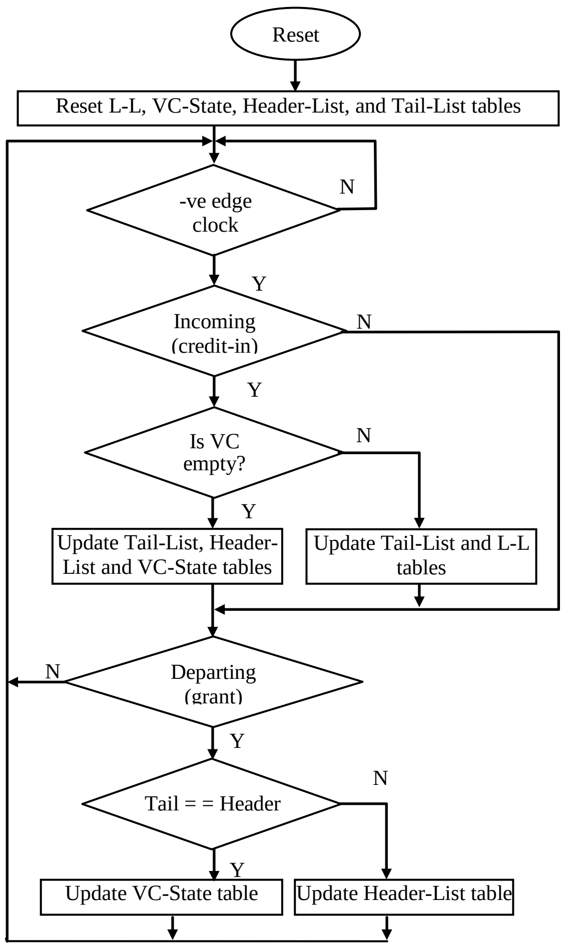
Figure 2.13: Timing Diagram of a Pipeline Communication inside a LLD Input-Port.

2.4.4 Slot-State Process

The *Slot-State* table keeps a record of the occupied slots in the central buffer by maintaining a Boolean flag for each slot in the *Slot-State* table. The flowchart of Figure 2.14a illustrates the process of recording the states in a *Slot-State* table. When a flit occupies a slot in the central buffer, the corresponding bit for that slot is set. When a flit leaves the slot, the corresponding bit of the slot is reset in the *Slot-State* table. The decoder module that decodes the content of the *Slot-State* table is shown in Figure 2.15a. The decoder generates the *write-pointer* signal and it is connected to the Address-Write port of the central buffer. The decoder points to the first unoccupied slot of the buffer.

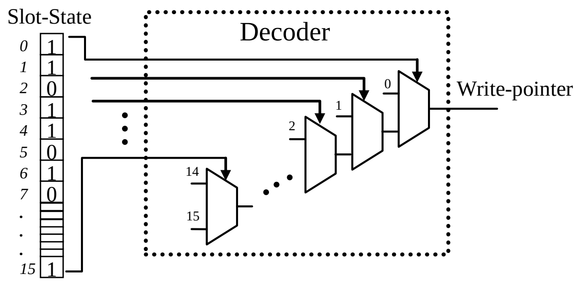

Reset
Reset Slot-State table
-ve edge clock
N
Y
Incoming (credit-in)
N
Y
Set related entry Slot-State[i] = 1 where i = write-pointer
N
Departing (grant)
Y
Reset related entry Slot-State[i] = 0 where i = read-pointer

(a) Slot-state

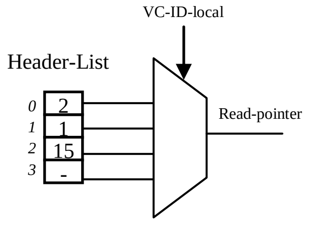


(b) Flit arrival and departure

Figure 2.14: Updating of LLD tables



(a) 4-bit decoder for the Write-pointer



(b) 4-bit Read-pointer

Figure 2.15: LLD Read and Write Pointers

2.4.5 Flit Arrival and Departure Process

We briefly describe the arrival and departure of data flits and updating of LLD tables. The updating mechanism of the LLD tables can be illustrated by the two processes of Figure 2.14. These two processes are sensitive to the negative edge of the router clock. The flit arrival and departure is detected by *credit-in* and *grant* signals, respectively. Assume that a VC (e.g. VC3) is empty and ready to accept data. Upon the arrival of a flit for VC3, three things happen simultaneously. First of all, the corresponding bit of VC3 is set in the *VC-State* table indicating that VC3 is occupied. Then the content of the *write-pointer* is stored in the *Tail-List* and *Header-List* tables. Finally, the corresponding bit is set in the *Slot-State* table. The *write-pointer* is then updated that points the next free slot for the incoming flit. When another incoming flit tries to move into the same VC (VC3), three things occur at the negative clock edge. The content of the *write-pointer* is stored in a location of the L-L table where the *Tail-List* table points to it. Then the *write-pointer* content is stored in the *Tail-List* table. Finally, the *Slot-State* table is updated, which leads to the updating of *write-pointer*.

When a flit exits from a VC (e.g. VC3), three types of events take place. If the Header and Tail addresses are the same (the last flit), the corresponding bit is reset in the *VC-State* indicating an empty VC3. In case the Header and Tail addresses are not same, the location of the L-L table is identified by the *Header-List* table, and it is stored as the new header address of VC3 in the *Header-List* table. Finally, the corresponding bit of the *Slot-State* table is reset, which causes the *write-pointer* to be updated.

As the flit arrival and departure occurs at two different locations of the input-port buffer, there is no storage conflict in the *Slot-State* table. When the Tail and the Header addresses are same, the *VC-State* table is updated (i.e. when the departing flit is the last flit of VC). Therefore, if the arrival and departure are for the same VC then the Header value is already updated. It means that the Tail and Header addresses are not the same and there will be no table updating at the flit departure. If the flit arrival and departure is for two different VCs, there would be no storage conflict due to independency of different cells of the *Header-List* table.

2.4.6 VC-block Signal

As illustrated earlier in Figures 2.13 and 2.14, all the tables are updated at the negative edge of the router clock that issues *request* signals and the *read-pointer* address. These signals are stable for arbitration at the positive clock edge (when *request* signals are checked by the arbiter module). When any one of the *request* signals is high, the arbiter reads the information of the requested flit for arbitration. If the requested output of a flit is free, the arbiter issues a *grant* signal. The *grant* signal causes the flit to exit the router at the positive clock edge. If the requested output is blocked, the arbiter issues a block signal (*VC-block*) that causes the *VC-Selector* to select any other available VC for service as illustrated in Figure 2.16.

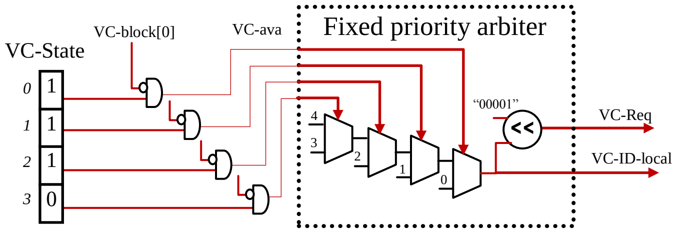


Figure 2.16: VC-Selector Request Logic

2.4.7 VC-Selector Module

The *VC-Selector* module in the router input-port selects a VC for arbitration. It issues the request signal, *VC-req* and is used to generate the *read-pointer* address as illustrated in Figures 2.16 and 2.15b. The *VC-Selector* module has a logic circuit that operates on the contents of *VC-State* table and creates VC availability signal (*VC-ava*). In fact, the *VC-block* signal is reversed and ANDed with its corresponding bit in the *VC-State* as illustrated in Figure 2.16. For example, if the VC0 is blocked then *VC-block*[0] signal is high to prevent the selection of VC0. The *VC-Selector* module selects a VC depending on the *VC-ava* signals and it consists of a logic circuit based on the following logic code.

```
if (VC-ST[0]&& !VC-block[0])  VC-req =1; // VC0 request
else if (VC-ST[1] && !VC-block[1])  VC-req =2; // VC1 request
      else if (VC- ST[2] && !VC-block[2]) VC-req =4;
            else if (VC- ST[3] && !VC-block[3]) VC-req =8;
                  else  VC-req =0;  // no request
```

The above code illustrates a deterministic scheduling policy that establishes a first priority for VC0, 2nd priority for VC1, and so on. *VC-ID*-local signals select a free or available VC for arbitration. In fact, the Header address of a VC is selected as the *read-pointer* as shown in Figure 2.15b.

2.4.8 Buffer-full Module or VC-full Module

In the traditional DAMQ scheme, a VC can occupy the entire input-port buffer space. Therefore, a *Buffer-full* signal is enough to represent the state of an input-port buffer. The *Buffer-full* signal is used to close the input-port. When the input-port buffer is full, all the cells of the *Slot-State* table are high. Therefore, the logical ANDing of the *Slot-State* cell values can issue the *Buffer-full* signal as shown in Figure 2.17a.

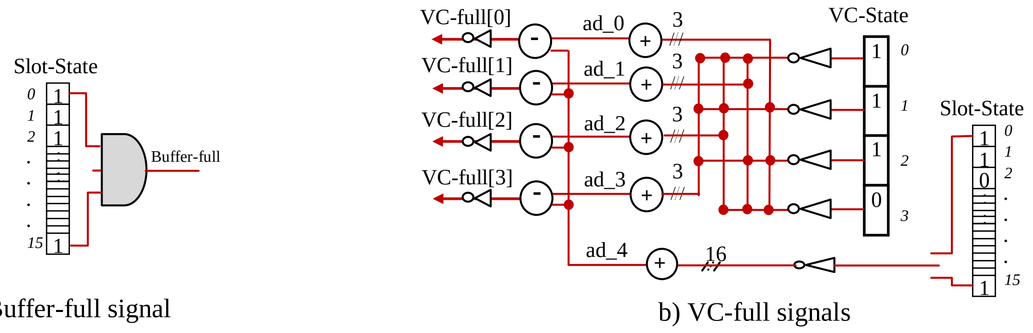


Figure 2.17: Input-Port Buffer-Full and VC-Full Modules

In case that the mechanism has control over the size of VCs, a credit signal (*VC-full*) is required for each VC instead *Buffer-full* signal. We present the scheme illustrated in Figure 2.17b where one slot is reserved for each VC. Assume that S0 to S15 represent the cell states of *Slot-State* table, and V0 to V3 represent the cell states of *VC-State* table. The state of *VC-full* signal is determined according to the logic equation as follows. VC-full[i]=((S0+…+S15)-(V0+…+V3)-Vi) where i϶{0…3}. For example, if VC0, VC1 and VC2 have at least a flit and VC3 is empty, their associated adder outputs, *ad_0, ad_1, ad_2* and *ad_3* become 1, 1, 1 and 0 respectively. When at least a slot of buffer is empty, the associated adder output, *ad_4* becomes 1. Now the states of *VC-full*[0], *VC-full*[1], *VC-full*[2] and *VC-full*[3] become 1, 1, 1 and 0 that means only VC3 is free.

2.5 Buffering Organization Approach

A number of researchers have focused on the design and organization of routers buffers due to its tight relation with the NoC power, performance and area. Some of them have used a complex architecture as compared to conventional VC-based router architecture, or they are efficient under certain NoC configurations or data flow circumstances [19, 34, 35, 36]. Some of these research works are discussed in this section.

An NoC router architecture called CUTBUF has been presented by Zoni *et al.* [34].The approach dynamically assigns VCs to input-port depending on the actual input-port load and reusing each queue by packets of different types. The approach significantly reduces the number of physical buffers in routers, thus saving area and power without decreasing NoC performance. They have assumed that a VC in conventional VC protocol can be re-allocated to a new packet only if the tail of its last allocated packet is sent out, i.e. it is empty. However, a reserved VC in CUTBUF protocol is released when either the tail flit traverses the same pipeline router stage, or when the related packet gets blocked. In other words, to increase buffer utilization and preventing HOL blocking in CUTBUF communication, a VC can be re-allocated to a new packet under two conditions. First, the tail of its last allocated packet is sent out, i.e. it is empty. Secondly, the packet that is stored in the queue buffer is guaranteed to traverse the switch in a fixed number of cycles. In such a way, the newly allocated packet is guaranteed not to be blocked because the previous one is guaranteed to traverse the switch. To implement above mechanism, the following conditions are checked to let a new packet to allocate a no empty VC. Once a packet has been granted by a router and its tail flit is stored in the input buffer VC, and the downstream router has enough credits to store all the flits of the VC. As mentioned before, these conditions are required for buffer reuse i.e. assigning a no empty VC to a new packet. A drawback of approach can be the protocol constrain assumed for conventional VC protocol. In fact, they assume that a VC in conventional VC protocol can be re-allocated to a new packet only if it is empty. Therefore, we expect that the results of approach are created based on this protocol constrain. Moreover, the efficiency of NoC without aforementioned protocol constrain has not been investigated in the research.

This part of conventional NoC protocol has also been investigated in an approach called Packet-Based Virtual Channel (PBVC) [35]. A VC in PBVC scheme is reserved when a packet enters the router and released when the packet leaves. A VC will hold the flits of only one packet at a time that subsequently removes the Head-of-Line (HoL) blocking. PBVC technique is more efficient in dynamically allocated multi-queue (DAMQ) schemes where an input or output port employs a centralized buffer whose slots are dynamically allocated to VCs. The experimental results show that in the case of HoL specific traffic, the average latency and throughput are improved for the PBVC approach as compared to conventional DAMQ-based NoC.

Yung-Chou and Yarsun have presented a new DAMQ-based buffer organization called DAMQ-MP that can accommodate multiple packets more than the number of virtual channels [19]. The approach can solve certain data transmission issues under some circumstances, such as heavy network congestion or short packets to improve the performance. To implement the DAMQ-MP, two assumptions are applied to the conventional DAMQ configurations as follows. First, a virtual channel only holds one packet each time for easy control and preventing HoL blockings. In other words, if all virtual channels contain packets whose tail flits have entered but not left yet, the remaining free buffer resources are wasted. The second assumption is to presume a long interval between the departure of tail flit belonging to the current packet and the header flit belonging to the next packet. By considering the above two assumptions, they have introduced the DAMQ-MP data flow as follows.

Whenever the tail flit of a packet enters the input buffer via one of the VC, this packet releases the virtual channel by alerting a notification signal to the upstream router. Therefore, a new packet from the upstream router can be sent out and enters this free VC. In fact, the new packet does not need to wait until the tail flit of previous packet gets out of the VC. This approach can saves a lot of times, especially in case that the routing computation for the arriving packets is high (see the second assumption). The aforementioned two assumptions and above data flow organization leads the DAMQ-MP to be efficient in the cases of short packets, large buffer capacity, heavy congested traffic (including saturated network), and small number of virtual channels. As one may notice, the DAMQ-MP approach is in reverse conclusion to the PBVC approach. The DAMQ-MP approach lets a VC to accept new packet when the VC is not empty, but the PBVC approach prevents a VC to accept new packet when the VC is not empty. In fact, both approaches are efficient under different NoC schemes and traffic patterns.

A router architecture, RoShaQ that allows sharing multiple buffer queues has been presented by Tran and Baas [36]. In this approach, each input-port allocates one buffer queue and shares all remaining queues. The router architecture maximizes buffer utilization by allowing the sharing multiple buffer queues among input-ports. Sharing queues, in fact, makes using buffers more efficient by reducing packet stall times at input-ports. The RoShaQ is able to achieve higher throughput when the network load is heavy. On the other side, at light traffic load, RoShaQ router achieves low latency by allowing packets to effectively bypass these shared queues and reducing zero-load packet latency. In conclusion, the proposed router achieves higher performance in both cases i.e. when the traffic load becomes heavy or at low-load traffic. An RoShaQ NoC is deadlock-free due to the following reasons. At light load, packets normally bypass shared queues, so the RoShaQ acts as a wormhole router hence the network is deadlock-free. At heavy load, if a packet cannot win the output-port, it is allowed to write only to a shared queue which is empty or contains packets having the same output-port. Clearly, in this case the RoShaQ acts as an output-buffered router which is also shown to be deadlock-free. A drawback of this approach is that the approach has achieved a weak contribution to NoC. In fact, the approach combines two switching mechanisms: VC-based and simple wormhole to improve the performance. When the first packet enters to an input-port, it is serviced through wormhole switching without involving VC pipelines. However, when the second packet enters (assume the first packet is still there), it is serviced through VC-based switching with involving VC pipelines. Therefore, the RoShaQ routers should accommodate both circuits related to wormhole switching and VC-based switching i.e. more hardware. Moreover, an extra circuit is needed to detect wormhole packets from VC-based packets. Therefore, the RoShaQ router architectures become more complex and consume more hardware as compared to conventional VC-based routers. The performance improvement of RoShaQ is due to higher hardware overhead not due to efficient architecture.

To eliminate buffer cost, Michelogiannakis and Dally have proposed an Elastic Buffer (EB) flow control [37]. Flits advance to the next EB using a ready-valid handshake. An EB asserts its ready signal routed upstream to indicate that it has at least one free storage slot. Furthermore, an EB asserts its valid signal routed downstream to indicate that it is driving a valid flit. When ready and valid are asserted between two EBs at a rising clock edge, a flit has advanced. This timing requires at least two storage slots per EB to avoid unnecessary pipeline bubbles as illustrated in Figure 2.18. In other words, an EB is a FIFO with two storage locations. EB channels use many consecutive EBs to form a distributed FIFO. Without virtual channels, deadlock prevention is achieved by duplicating physical channels.

In other words, EB is a primitive and simplified form of NoC buffering, which can be easily integrated in a plug-and-play manner at the inputs and outputs of the routers as well as on the network links to act as a buffered repeater. EB assumes only one form of handshake on each network channel. The handshake cannot distinguish between different flows, thus making the EB operation serial in nature. This feature prevents the interleaving of packets and the isolation of traffic flows, while it complicates deadlock prevention. Due to this limitation, direct support for VCs is abandoned and replaced by multiple physical networks or implemented via complex and non-scalable hybrid EB/VC buffering architectures [37]. Hybrid routers operate as EB routers in the common case, and as VC routers when necessary. However, hybrid EB/VC technique removes the basic property of the EBs to act as elements that can be placed seamlessly anywhere in the NoC.

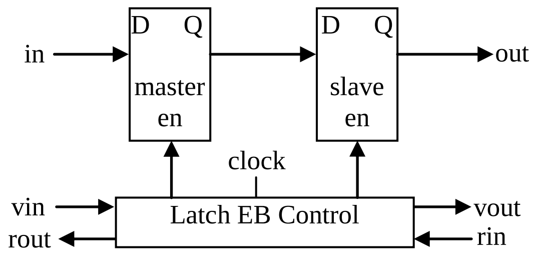


Figure 2.18: Any EB architecture derived for edge-triggered flip-flops can also be implemented with latches[55].

Seitanidis *et al*. have extended EB architectures to support multiple virtual channels [38]. They called their EB architecture, ElastiStore. The ElastiStore approach minimizes buffering requirements without sacrificing performance and without introducing any dependencies between VCs, thus ensuring deadlock-free operation. It uses just one register per VC and a shared buffer sized large enough to only cover the round-trip time that appears either on the NoC links or due to the internal pipeline of the NoC routers as illustrated in Figure 2.19.

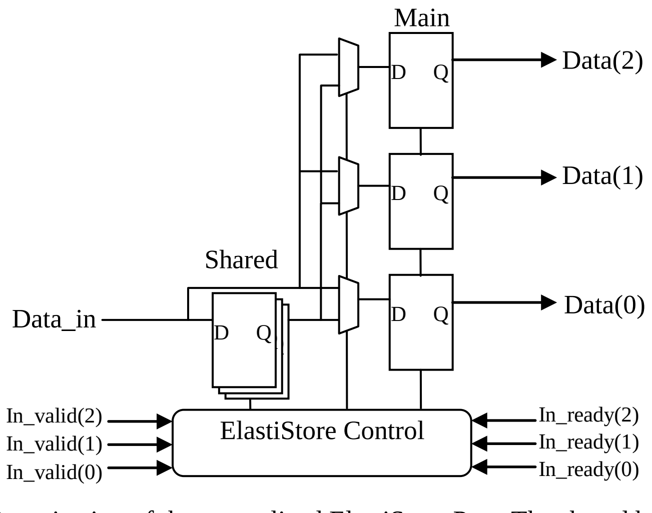


Figure 2.19: Organization of the generalized ElastiStore Port. The shared buffer consists of as many buffer slots as required to cover the round-trip time of the flow-control signals [56].

The integration of ElastiStore scheme in an NoC router is illustrated in Figure 2.20. This integration enables the design of efficient architectures, which offer the same performance as baseline VC-based routers, but at a significantly lower cost.

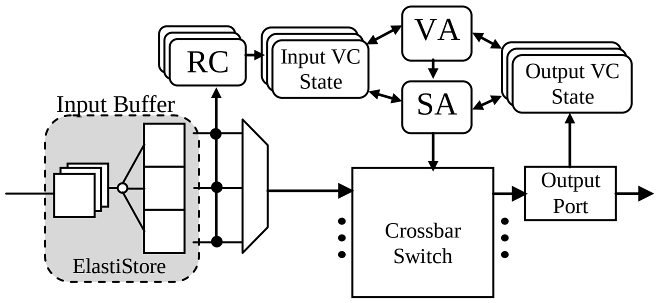


Figure 2.20: Integration of ElastiStore in NoC routers. ElastiStore modules can be integrated at the inputs and at the outputs of a router. In general, ElastiStores can be placed seamlessly and in a plug-and-play manner everywhere within the NoC [56].

In fact, this approach represents the DAMQ architecture with a reserved slot per each VC, since it has been stated in [38] that ElastiStore architecture rely on fairly complex logic to keep track of the location of flits within the unified buffer (see buffers illustrated in Figure 2.19). The ElastiStore-based NoC results are only compared with those of conventional VC-based NoC. For sake of fair evaluation, this approach could be also compared with a DAMQ NoC.

2.6 Heterogeneous Router Architectures

A number of researchers have focused on the design and organization of routers that have direct impact on the NoC power, performance and area [39, 40, 41]. Some of these research works are discussed in this section. A hybrid switching mechanism, Virtual Circuit Switching (VCS), is proposed by Jiang *et al*. [39]. The approach intermingles the Circuit Switching (CS) and Packet Switching (PS) to obtain low latency and power consumption in NoCs. They have also proposed a path allocation algorithm to determine VCS connections and CS connections in a mesh-connected NoC. VCs in the VCS approach are exploited to form a number of VC connections by storing the interconnect information in routers. Flits can directly traverse the routers by using only Switch Traversal (ST) stage (see Section 1.6.1). The main advantage of VCS approach is that it can have a similar router pipeline with circuit switching, and can have multiple VCS connections to share a common physical channel. In this hybrid scheme, VCS connections cooperate with PS connections and CS. Once flits on CS or VCS connections arrive at routers, crossbar switches are immediately configured so that the CS or VCS flits can bypass directly to the ST stage. When there is no CS or VCS flit, the corresponding ports of crossbar switches are released to PS connections. Figure 2.21 shows VCS, CS, and PS connections of the VCS scheme. The VCs of routers in VCS connections are preconfigured in such a way that they are only connected to the particular downstream router VCs. Crossbar switches of routers are preconfigured during the Switch Allocator (SA) stage before VCS flits require passing through.

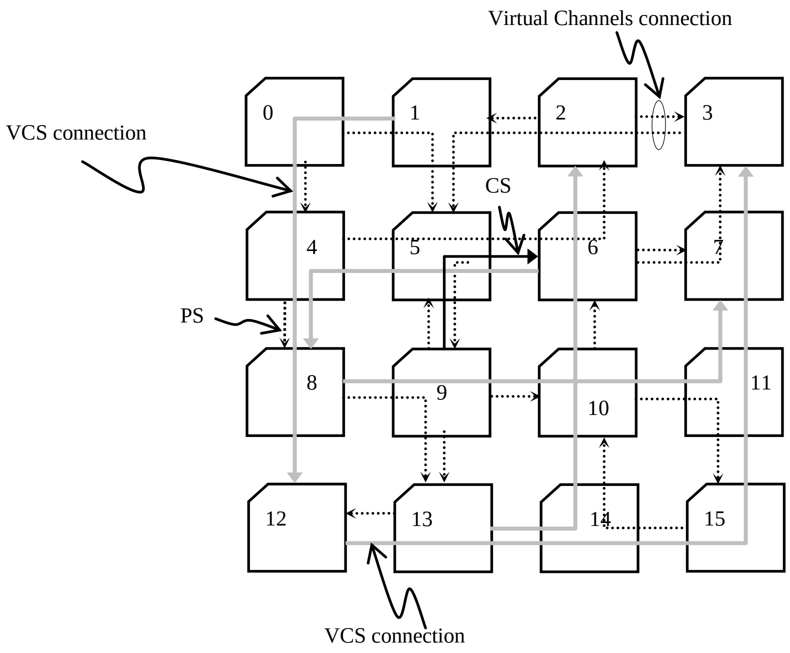


Figure 2.21: Proposed Hybrid Scheme in A 4 × 4 Mesh with two VCs per Input-Port [47].

As VCS connections are established over VCs, a physical channel can be shared by *n* VCS connections at most where *n* is the number of VCs. Other communications competing for the same physical channel must follow the packet switching, e.g. the communication from node 4 to 8 shown in Figure 2.21. There are two drawbacks regarding of the VCS approach. Firstly the VCS routers have to the circuitry required for both packet and circuit switching mechanisms. Moreover, an extra hardware is needed to detect VCS packets from PS packets. In this way, the VCS router architectures are complex and required more hardware as compared to conventional VC-based wormhole routers. The second drawback is that despite the higher hardware of VCS, the approach is not efficient for varying communication application. On the other hand, it is efficient for deterministic communications e.g. the communication in application-specific NoCs.

In application-specific NoCs, the routing connections are determined and fixed, and the router can be preconfigured for CS or VCS connections in advance, so that the CS or VCS packets can pass through their specific routes to reach their destinations. However, our proposed NoC routers in Chapter 3 and 4 are efficient for any type of NoC communications. Moreover, they improve the performance while their architectures are simpler and accommodate lower hardware overhead.

Another heterogeneous NoC router architecture has been introduced by Ben-Itzhak *et al.* [40]. They exploit a shared-buffer technique in order to handle the heterogeneity of NoC link bandwidths. Their approach reduces the number of shared-buffers required for a conflict free router without affecting the performance. Reducing the number of required shared-buffers also reduce the crossbar size and overall it will reduce the chip area and power consumption. The heterogeneous router architecture supports different link capacities and different number of VCs for each unidirectional link while keeping the router frequency fixed as illustrated in Figure 2.22.

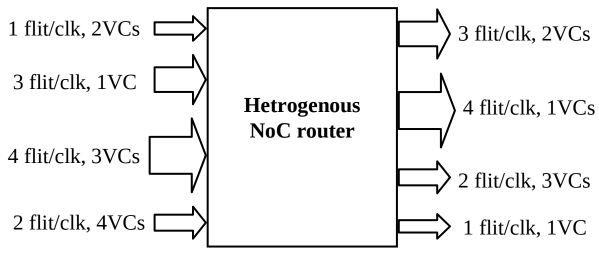


Figure 2.22: Heterogeneous NoC Router Example [48].

The heterogeneous architecture utilizes serial-to-parallel converters in order to store incoming flits to different input buffer slots at each link clock cycle, and parallel-to-serial converters can be used in order to transmit several flits in Time-Division Multiplexing (TDM) fashion depending on the link frequency. In fact, the shared-buffer technique presented by Ben-Itzhak *et al.* optimizes the number of shared-buffers related to arrival and departure conflicts discussed by Ramanujam *et al.* in [41]. The drawbacks of VCS approach discussed earlier can be considered for the heterogeneous approach. In other words, the heterogeneous router architecture is tailored for a specific type of NoC communication. However, NoC has emerged as a network with scalable, reusable and global communication architecture to address the SoC design challenges. The NoC features enable it to be easily expandable and more important to provide services for a variety of SoC communications. The standard heterogeneous feature of the approach violates the scalability and reusability of NoC, and it needs more research and investigation.

2.7 Round Robin Arbiter

In digital system design, arbiters are used to allocate and access shared resources. Whenever a resource, such as a buffer, channel or a switch-port is shared, an arbiter is required to assign the access to the resource at a particular time. The most common usage of arbiters is the shared-bus arbitration of a bus-based system where multiple master modules can initiate their transactions. The modules must be arbitrated for access to the bus before initiating a transaction. In this dissertation, we investigate the arbiters used in NoC systems. Figure 2.23 illustrates a wormhole *v*-VC router where the Router Arbiter module includes a Switch Allocator module consisting of two sets of simpler arbiter. A simpler arbiter arbitrates among a group of requesters for a single resource as illustrated in form of a symbol for an *n*-input in the right side of Figure 2.23.

The arbiter accepts *n* requests ($r0, r1, . . ., r_{n-1}$), arbitrates among the asserted request lines, and selects an *ri* for service, and then asserts the corresponding grant line, *gi*. For example, assume the arbitration for the output-port of a crossbar switch among a set of requests from the VCs of some input-ports. The input-port VCs that have flits will issue request signals for having access to one of the desired output-port. Assume, there are 5 VCs and VCs 0, 2, and 4 assert their request lines, *r0*, *r2*, and *r4* respectively. The arbiter will then arbitrate and select one of these VCs for assigning the desired output-port. Assume the grant of VC2 (i.e. *g2*) is asserted. VCs 0 and 4 lose the arbitration and must hold their requests active until they receive the grant signal for their output-ports.

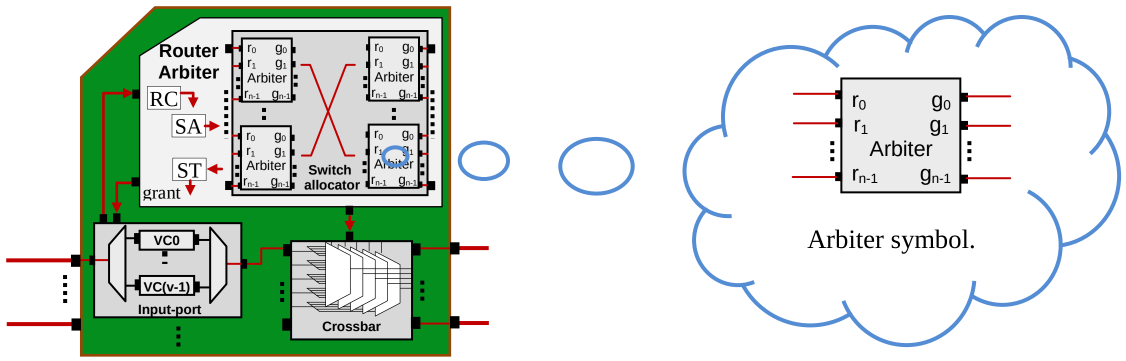


Figure 2.23: A wormhole *v*-VC router, the Switch allocator consists of two sets of simpler arbiter.

2.7.1 Conventional Arbiter Design

Arbiters can be categorized in terms of fairness (weak, strong or FIFO) arbiters [2, 42]. In a weak fairness arbiter, every request is eventually granted. The requests of a strong fairness arbiter will be granted equally often. The requests of FIFO fairness are granted in a first come first served basis. Moreover, arbiters in terms of priority can be grouped in two fixed and variable architectures. For a fixed priority arbiter, the priority of requests is established in a linear order. Figure 2.24 illustrates a 4-input fixed-priority arbiter where r0 has the highest and *r3* has the least priority [2]. The architecture can be expanded to *n*-input arbiter where for each middle request, there is an arbiter cell consisting of two ANDs and an Inverter. The first and last arbiter cells can be simplified (see Figure 2.24). For each request input *ri*, there is a carry input *ci,* a grant output *gi*, and a carry output, *ci+1* where *i* ϵ {*0, 1... n-1*}. Therefore, a low level *ci* indicates that at least one of requests from r0 to *ri-1* was has been asserted. Moreover, in case that the request *ri* and carry *ci* are high, the grant, *gi* is set, and all the following grants i.e. *gi+1* to *gn-1* will become reset. It is obvious that the critical path of the circuit is from the first request, *r0* to the last grant, gn-1 due to propagation of carry from head to the tail of the arbiter. Fixed priority arbiters provide weak fairness arbitration because when a request is continuously asserted, none of its following requests will ever be served.

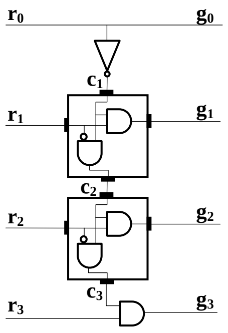


Figure 2.24: A 4-Input Fixed Priority Arbiter Architecture.

In order to have a fair iterative arbiter, we can use a variable priority arbiter as illustrated in Figure 2.25. An OR gate and a priority input signal, *pi* is added to each cell of the fixed priority arbiter shown in Figure 2.24. When *p1* is set, its corresponding request, *r1* has high priority and the priority decreases from that point cyclically around the circular carry chain.

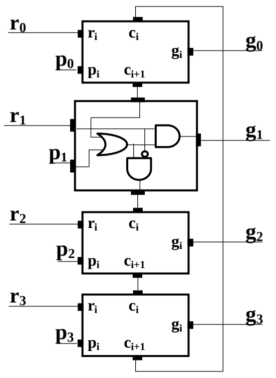


Figure 2.25: A 4-Input Variable Arbiter Architecture.

Now we can create a fair iterative arbiter by changing its priority from cycle to cycle. In an *n*-input arbiter, if the grant, *gi* (where *i* ϵ {*0, 1, ... n-1*}) is connected to the next priority vector *pi+1*, a Round Robin (RoR) arbiter is created. Figure 2.26 illustrates a 4-input RoR arbiter. If a grant, *g1* becomes high at the current cycle, it causes *p1* to be set high on the next clock cycle. This leads the request, *r2* to become the highest priority at the next cycle, where the request, *r1* becomes the lowest priority. For the sake of simplicity, we assume that the arbitration cycle takes one clock cycle in all the architectures describe in this dissertation.

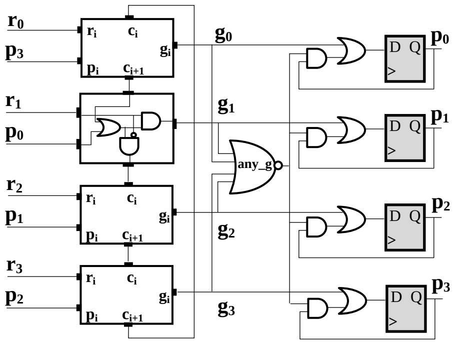


Figure 2.26: A 4-Input RoR Arbiter Architecture.

The functionality of a round-robin arbiter can be explained as a request that is just granted will have the lowest priority on the next arbitration cycle [2]. The round robin arbiters are simple, easy to implement, and starvation free. When the input requests are large in numbers, the structure of round robin arbiter grows that leads to large chip area, higher power consumption, and critical path delay. In an NoC design, the critical path delay of arbiter usually dominates among the critical path delays of input-port and crossbar switch due to the architectural complexity of arbiter as compared to those of port and crossbar switch. Therefore, the arbiter circuit determines the maximum frequency (or the speed), f_{max} of an NoC router. The critical impact of arbiter on the performance of the NoC system and the characteristic behaviour of round robin architectures have created a lot of interest of NoC researchers as we will discuss some of them in the following section.

2.7.2 Some Well-Known RR Arbiters

The architecture of a popular Matrix round robin arbiter is presented by Dally and Towles [2]. A 4-input Matrix arbiter architecture is shown in Figure 2.27. It implements a least recently served priority scheme where a request, *ri* wins arbitration. It resets the bits of row *i* and sets the bits of column *i* to make itself the lowest priority where $i \in \{0,..3\}$. The Matrix arbiter is claimed to be useful for small number of inputs as it is fast, economical, and performs strong fairness arbitration. However, no evaluation is presented. Fu and Ling evaluated and compared the RoR and Matrix arbiters in terms of resource, performance and power consumption for an FPGA platform [43]. They concluded that the Matrix arbiter consumes more resource, same power but can process data more quickly than the RoR arbiter.

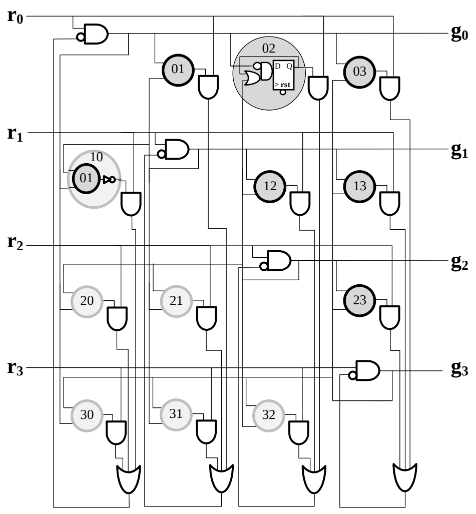


Figure 2.27: A 4-Input Matrix arbiter.

Zheng and Yang proposed a Parallel Round Robin Arbiter (PRRA) based on a simple binary search algorithm as illustrated for a 4-input PRRA in Figure 2.28 [44]. They further proposed an Improved PRRA (IPRRA) design where the output signals, *gL* and *gR* of PRRA are disconnected and directly ANDed with grant signals to generate new grant signals as shown in Figure 2.29. The IPRRA reduces the timing of PRRA significantly.

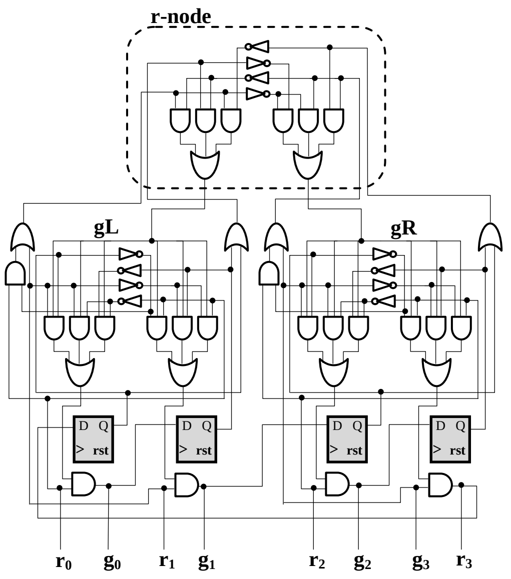


Figure 2.28: A 4-Input PRRA Architecture.

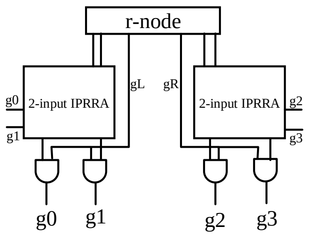


Figure 2.29: A 4-Input IPRRA Architecture.

A High speed and Decentralized Round robin Arbiter (HDRA) has been presented by Lee et al., which is illustrated in Figure 2.30 [45]. Each circuit enclosed with dash circle represents a filter circuit whose main components are a D flip-flop and a multiplexer. The filter circuit filters out the input without request or the one with request that has already been granted at that arbitration cycle. The un-filtered inputs with their requests participate in the arbitration again next cycle by setting its corresponding D-type flip-flops to 0 that are done by enabling the *ack* signals from higher lower level. The HDRA arbiter will reset itself asynchronously by the input *self_rst* from the root. The *sys_rst* indicates the system reset signal and is used initially before each arbitration cycle for all requests. A 4-input HDRA arbiter has a simpler circuit than a higher input HDRA architecture because the *act, rnext* and *self-rst* are connected together. The HDRA architecture is used later in some other works e.g. in the router model of simulation framework implemented by Guderian *et al*. [46]. In Chapter 5, we introduce a novel RR arbiter that is simpler, faster and consume lower hardware overhead as compared to the aforementioned arbiters (i.e. RoR [2, 43], Matrix [2, 43], HDRA [45], PPRA [44] and IPRRA [44]).

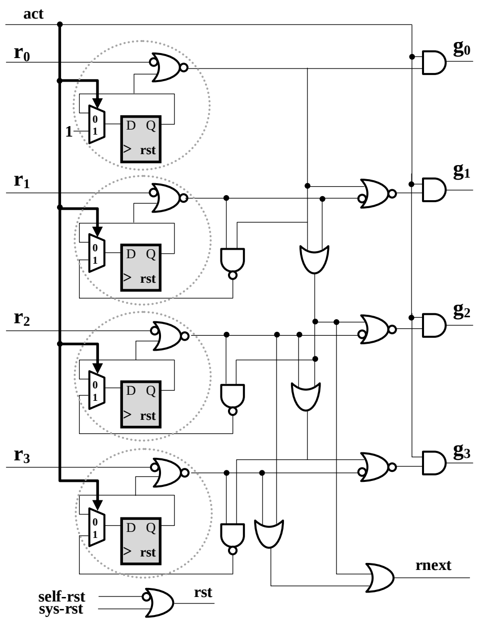


Figure 2.30: A 4-Input HDRA Architecture.

Chapter 3

Statically Adaptive Multi-FIFO Buffer Organization

In this chapter, we present a new technique for efficient flow control and adaptive VC allocation for on-chip applications. We implement an adaptive virtual channel flow control to demonstrate higher buffer utilization, improved message latency and NoC throughput. In terms of hardware, we present an NoC architecture leading to lower size and power consumption due to SRAM-based buffer sharing. In terms of pipeline stages, we show that our mechanism is similar to the conventional static VC mechanism, so we can benefit of fastest arrival and departure of data. Our contribution is described as follows. The basic concept and communication of our approach while they are compared with a conventional VC-based architecture are described in Sections 3.1 and 3.2 respectively. In Section 3.3, we discuss that our model is similar to the conventional static model in term of functionality, but it can provide advanced adaptivity as described in Section 3.4. The hardware modeling of our approach is discussed and evaluated in Section 3.5 while the adaptivity of our approach is evaluated by various experimental results in Section 3.6. Finally, novel features and summary of the approach are listed in Sections 3.7 and 3.8.

3.1 Static Multi-FIFO (SMF) Buffer Architecture

In a typical SRAM based FIFO, two pointers point to the address of memory where the data is read or written [23, 47, 48]. Figure 3.1 shows a simple scheme of an SRAM-based FIFO. For a read operation, the *Read-Pointer* is incremented to point at the next slot (assume P1).

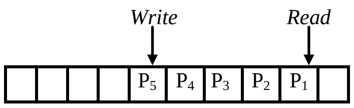


Figure 3.1: A Typical SRAM-Based FIFO

Consequently, the content of the slot (P1) appears at the output. In the case of a write event, the incoming data is stored at the location pointed by the *Write-pointer* (assume P5), and the *Write-Pointer* is incremented to the next empty slot. The difference between *Write-pointer* and *Read-pointer* determines whether the FIFO is full or empty. A buffer-full condition occurs when a write results in the difference of these pointers to zero. An empty condition occurs when the difference of two pointers become zero after a read. Both *Read-pointer* and *Write-pointer* increase circularly. This FIFO mechanism has enabled us to create a multi queue buffer by using an SRAM. In fact, by switching multiple read and write pointers to the address port of a SRAM, multiple FIFO buffers are created in a single SRAM. Figure 3.2 shows the architecture of a channel buffer (input-port) that employs our proposed Static Multi FIFO (SMF).

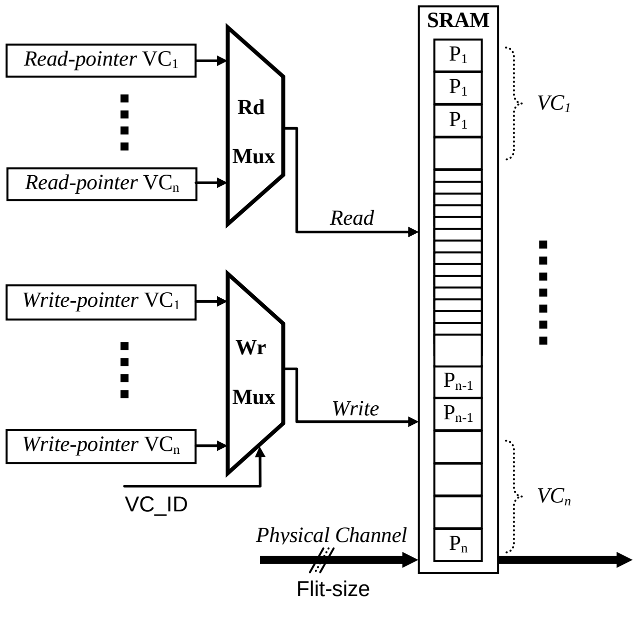


Figure 3.2: SMF Input-Port Architecture (*n*VC)

In order to investigate and evaluate our proposed architecture, a Conventional Virtual Channel (CVC) NoC router input-port architecture is considered for comparison and it is illustrated in Figure 3.3. In CVC architecture, each FIFO represents a VC, and the number of VCs will be equal to the number of FIFOs [1]. The number and the size of FIFOs (VCs) are constant in a CVC architecture. In the following sections, we present the details of CVC and SMF micro-architectures in an effort to illustrate both these mechanisms that are similar in terms of their functionality. However, the SMF architecture can easily accommodate the advanced adaptivity of the VCs.

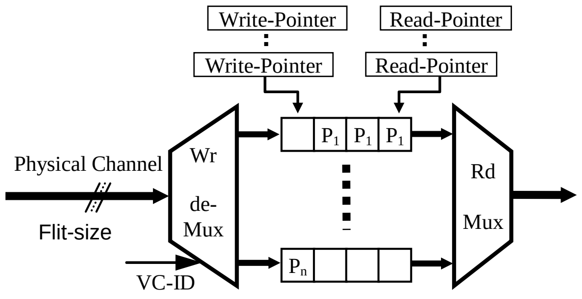


Figure 3.3: CVC Input-Port Architecture (*n*VC)

3.2 Communication in CVC and SMF Models

A VC-based wormhole communication in the CVC and SMF models are shown in Figures 3.4 and 3.5 respectively. We assume parallel communication in both models i.e. the width of physical channel is equal to the size of the flits. Moreover, each physical channel has four VCs and four packets are going to share a physical channel on a flit-by-flit basis. Furthermore, an NoC router must be able to de-multiplex the flits of packets at their entrance and direct them to their respective VC buffers. For this purpose, an identification signal (*VC-ID*) is sent before latching a flit to indicate its VC at the entrance of the input-port [49, 50]. As shown in Figures 3.4 and 3.5, the *VC-ID* signals can be transmitted on dedicated wires and connected to the select lines of the de-multiplexer. When a flit enters the input-port, it is directed to the VC location pointed by the *VC-ID.* In the case of SMF, these *VC-ID* signals select the *Write-pointer,* which should be connected to the write address port of an SRAM (see Figure 3.5). Therefore, a flit is stored in its dedicated VC area of SRAM on its arrival.

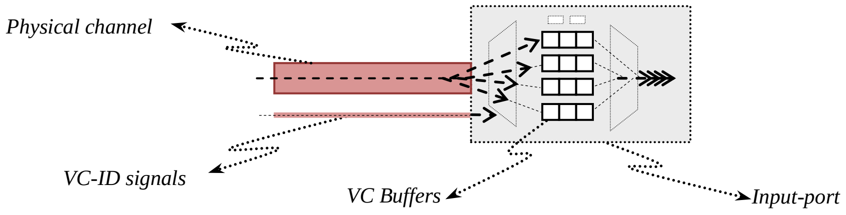


Figure 3.4: Conventional VC Flow Control

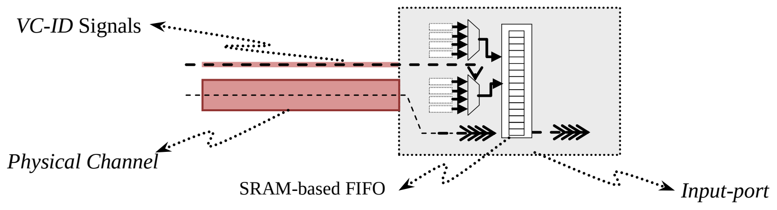


Figure 3.5: SMF Flow Control

Consider a packet P3 has a *VC-ID* of 3 that switches *Write-pointer VC3* as the writing address of SRAM as shown in Figures 3.6 and 3.7. The values of read and write pointers indicate that the VCs are occupied, full or empty. When a VC is full, the input-port issues an acknowledgment signal to the upstream router to stop any further flit transfer to it. In the case of no empty VC, the input channel issues a request signal to the arbiter. In case of multiple requests for an output channel, the request of one of the VCs can be processed by the arbiter. After arbitration, a grant signal (*grant)* is issued by the arbiter to latch the head flit of selected VC in the output register, *O-reg* in Figures 3.6 and 3.7. This leads the flit to move out of the input-port.

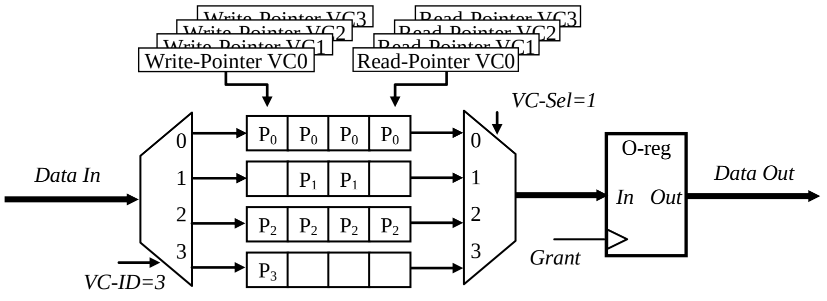


Figure 3.6: 4-VC CVC Buffer (VC3 Write and VC1 Read)

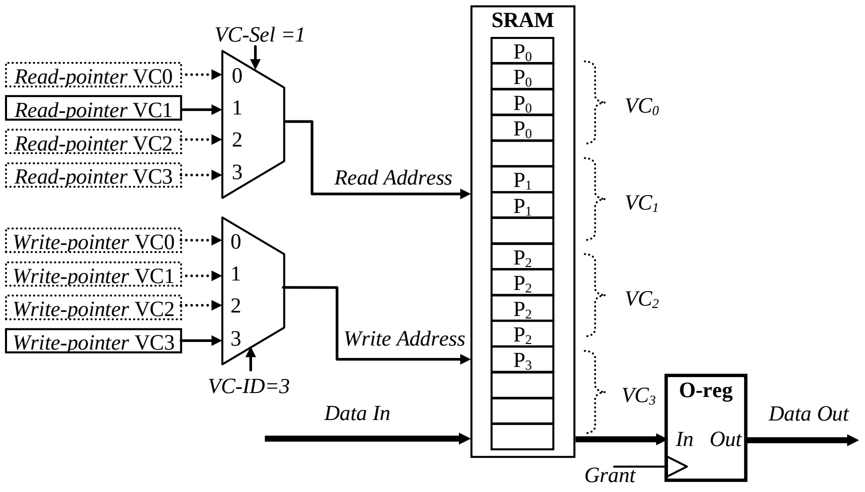


Figure 3.7: 4-VC SMF Buffer (VC3 Write and VC1 Read)

In Figure 3.6, the *Read-Pointer* of VC1 will be switched to the read address of SRAM when *VC-Sel* is active. In this way, the communication process indicates that the SMF model resembles the CVC model in terms of arrival and departure pipeline of flits.

3.3 Similarity of CVC and SMF during Contention

The structure of the *VC-Selector* module shown in Figure 3.8 illustrates the similarity in the flit departure process for the SMF and CVC models, especially in the case of NoC contention (blockage). The *VC-Selector* module is generally located in the arbiter, and it may have the same structure for both SMF and CVC models as shown in Figure 3.8. It works and behaves like a fixed priority arbiter. When the read and write pointers of a VC are different, the input-port issues a request signal (*Req*) to the arbiter. The request signals of all the VCs of a channel and the state of the corresponding outputs of those VCs (*VC-block*) are connected to the inputs of the *VC-Selector* module. The *VC-Selector* module contains a combinational logic circuit that selects a VC for arbitration in the arbiter module of router. The logic associated with the *Req* signals and the *VC-block* signals (active when the VC is blocked) create the VC availability signal (*VC-ava*). Actually, the *VC-block* signal is inversed and ANDed with its corresponding *Req* signal as shown in Figure 3.8. For example, if the requested output of VC0 is blocked then the *VC-block*0 signal is high to prevent the selection of VC0. A VC is selected depending on the *VC-ava* signal and a *VC-Sel* signal is issued. The *VC-Sel* is connected to the select lines of the read multiplexer of the input-port as shown in Figure 3.7. The output of the multiplexer contains the packet flit under arbitration.

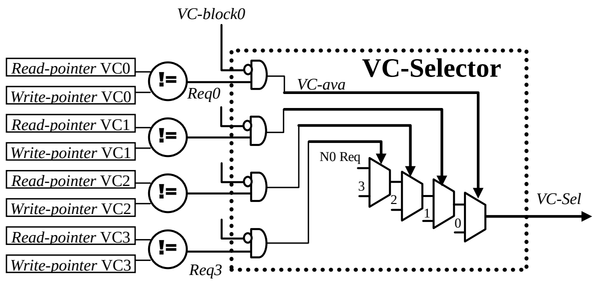


Figure 3.8: VC-Selector Circuit

Consider the non header flit of a packet (i.e. packet is already arbitrated), the packet information is already saved in the arbiter and its related *VC-block* contains the status of the associated output. In case that the flit is a header (new packet); its associated output will not be assigned. The circuit for finding an output is not sequential i.e. as soon as the *VC-Selector* selects a VC, its associated *VC-block* will have the state of the requested output. In a situation where the output is blocked, the *VC-Selector* selects another VC. One important feature of the dual-ported SRAM is that its reading function can be asynchronous (i.e., its output depends on the present value of the read address). Two important features, i.e. asynchronous SRAM reading and choosing the output-port allow a free packet to be selected for arbitration before the next clock cycle.

Figure 3.9 illustrates the selection function of a free VC. When the *VC-block* is active, there is either no free downstream VC (in the case of a new packet) or the associated downstream router VC becomes blocked. As previously mentioned, all of the circuits in the loop (VC-selector, reading part of SRAM, routing allocator, VC allocator, and OR gate) are not sequential. Moreover, the iteration in the loop leads to the selection of a free VC request. As already mentioned, we have presented the SMF and CVC architectures in detail to show that the arrival/departure function of a flit is similar for both SMF and CVC models especially in the case of a blockage in the NoC. The above discussion illustrates that the functional performances of both CVC and SMF mechanisms are the same. The reading delay from SRAM is bigger for SMF than that of CVC input-port buffers. However, the SMF architecture can provide advanced adaptivity for the VCs in a channel that will lead to higher performance, lower latency, improved buffer utilization, less power and lower IC area overhead as discussed in the following sections.

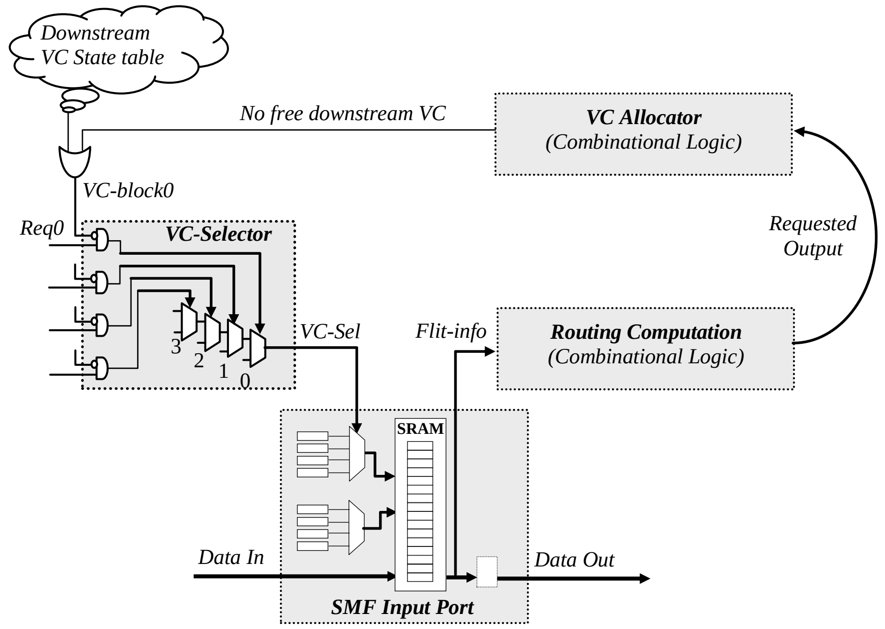


Figure 3.9: Selection Function of a Free Request

3.4 Statically Adaptive Multi FIFO

The SMF model includes multiple VCs that are created in an input-port by multiplexing their pointers. We have developed an adaptive feature for the SMF model by further multiplexing of pointers and controlling signals to create a Statically Adaptive Multi FIFO (SAMF) model. We call it statically adaptive as the VC buffers are static during a specific time of communication, and then they are subsequently adapted to the traffic demand. The SAMF model improves buffer utilization by closing redundant VCs and allocating their buffers to the active VCs. Higher VC buffer depth reduces contention and, subsequently improves performance. We incorporate some additional modules such as a *Mode-Selector* for each channel and a multiplexer for each VC, as well as employ the *VC-Full* (it is activated when a VC becomes full) signal to create an adaptive mechanism, as illustrated in Figure 3.10. The *Mode-Selector* module generates "adaptivity" signal (*adapt*) to cover *n* adaptivity modes where *n* is equal or less than the number of VCs per channel. Each adaptivity mode is defined by $m \times$VC; where *m* represents the number of VCs of a channel and $1 \leq m \leq n$. In the situation where *m* is less than *n*, the *Mode-Selector* issues the closing signal, *VC-close* to close the unused VCs. The *VC-close* signals are NORed with *VC-full* signal to represent the state of VCs as illustrated in Figure 3.10. In other words, when a VC is full or redundant within a channel (either *VC-full* or *VC-close* is set), the upstream router does not send flits for that VC.

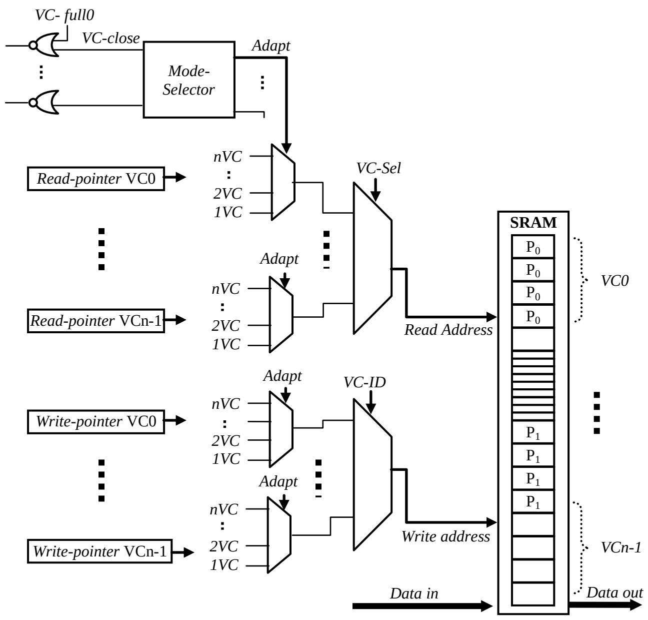


Figure 3.10: nVC SAMF Architecture.

To discuss the details of our approach, an SAMF architecture for four VCs and an SRAM of 16 slots (flits) is illustrated in Figure 3.11. In this example, the adaptivity modes are illustrated as follows. In case of a 4VC mode, each VC takes a quarter of the total SRAM slots. The 3VC mode activates VC0, VC1 and VC2 where VC0 has double the space than each of VC1 or VC2. The 2VC mode means that two VCs are accommodated in the SRAM (VC0 and VC1), where each VC contains half of the SRAM capacity. 1VC mode accommodates only one VC (VC0) in the SRAM. Table 3.1 shows the connections in the architecture to implement such adaptivity. The pointers can address the overall SRAM and their width is of four bits: *a*, *b*, *c* and *d*. *VC-Sel* has two bit signals *x* and *y*, and *VC-ID* also has two bit signals *p* and *q*. Different combinations of above bits (i.e. *a, b, c, d, x, y, p and q*) are used to specify different modes. For example, in 1VC mode all the output lines of pointers are connected to the address of the SRAM. In the case of a 2VC mode, three least significant bits of the pointers are used as least significant bits of the address ports. The least significant bits of *VC-Sel* and *VC-ID* are used for the most significant bit of read and write address ports of SRAM respectively. In 4VC mode, two least significant bits of pointers are used as the least significant bits of address ports, and two bits of *VC-Sel* and *VC-ID* are used for the two most significant bits of read and write address of SRAM.

Consider 4VC mode of Figure 3.11 and Table 3.1, where read address is ‘0011’ i.e. two least significant bits of *Read-Pointer VC0* are ‘11’ and two bits of *VC-Sel* are ‘00’. In 2VC mode, the

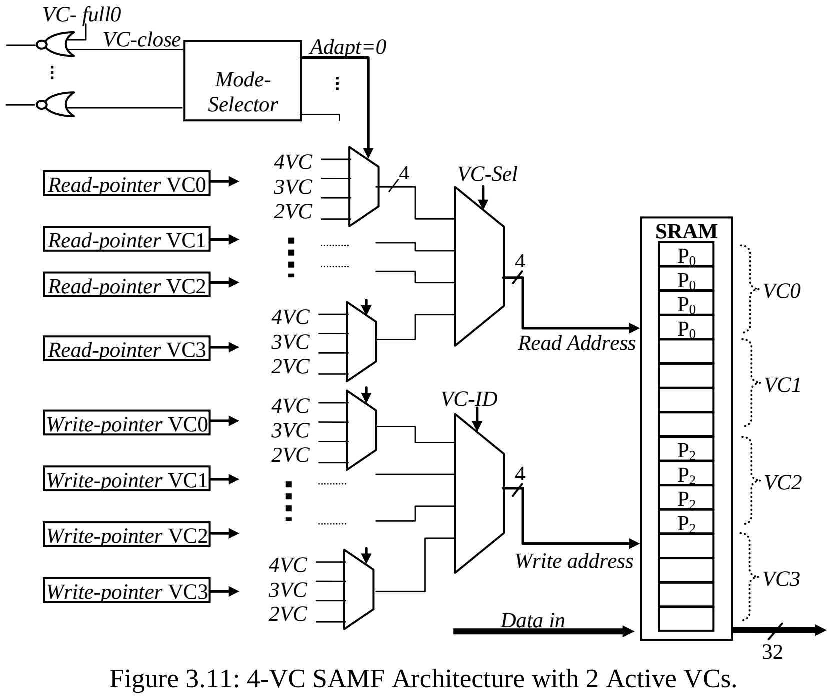


Figure 3.11: 4-VC SAMF Architecture with 2 Active VCs.

write address is ‘1100’ i.e. three least significant bits of *Write-Pointer VC1* are ‘100’ and the least significant bit of *VC-ID* is ‘1’. The *Mode-Selector* module is designed in such a way that different modes are deterministically applied based on the configuration of an NoC system. *Mode-Selector* module can also be designed in a complex form where different modes are dynamically applied according to NoC communication. We present our dynamic and deterministic *Mode-Selector* schemes in the following sections.

Table 3.1 Connection in SAMF Mode

Mode		***Adapt***	***VC-Sel***	**Pointers**	**Read add.**	***VC-ID***	**Write add.**	**VC-close**
4VC		00	*x,y*	*a,b,c,d*	*x,y,c,d*	*p,q*	*p,q,c,d*	*0000*
3VC	VC0	01	*x,y*	*a,b,c,d*	*y,b,c,d*	*p,q*	*q,b,c,d*	*1000*
	VC1,VC2		*x,y*	*a,b,c,d*	*x,y,c,d*	*p,q*	*p,q,c,d*	*1000*
2VC		10	*x,y*	*a,b,c,d*	*y,b,c,d*	*p,q*	*q,b,c,d*	*1100*
1VC		11	*x,y*	*a,b,c,d*	*a,b,c,d*	*p,q*	*a,b,c,d*	*1110*

3.4.1 Dynamic Mode Selector

A dynamic *Mode-Selector* module dynamically selects an optimum number of VCs according to the traffic demands in a channel and closes unused VCs. Figure 3.12 illustrates the process flow diagram of our dynamic *Mode-Selector*. The process is invoked when a packet enters an input-port. A variable, *Pkt-N* keeps a record of the number of new and varied packets. A new and varied packet means a packet with different source or destination ID. In such case, the variable *VC-N* is incremented by passing a varied packet through the associated channel. After a specific number of incoming packets that we call the *Selection-Factor*, the *VC-N* determines how many VCs are needed to handle the communication in an optimal fashion. The dynamic *Mode-Selector* process iteratively determines the number of channel VCs based on two factors: the channel traffic demand and the *Selection-Factor*. At the end of each iterative process all the variables reset. In fact, the traffic of a channel is measured in terms of the number of varied packets in each iteration of the process. According to this number, an appropriated adaptivity mode will apply. Switching from a higher mode (i.e., more VCs per channel) to a lower mode (i.e., less VCs per channel) is easy. It has no negative impact on communications and incurs little cost for an NoC system. We investigate this kind of switching in detail in the following sub-section. However, switching from a lower mode to a higher mode is complicated and slows NoC communication during mode switching. For example, assume a 4-VC input-port where the 1VC (VC0) is the current VC mode allowing data to be everywhere in the input-port buffer. In a situation where the 4VC becomes the new mode, the easiest way to accomplish mode switching is when the buffer becomes empty. This delay in empting the buffer slows communication at mode switching time.

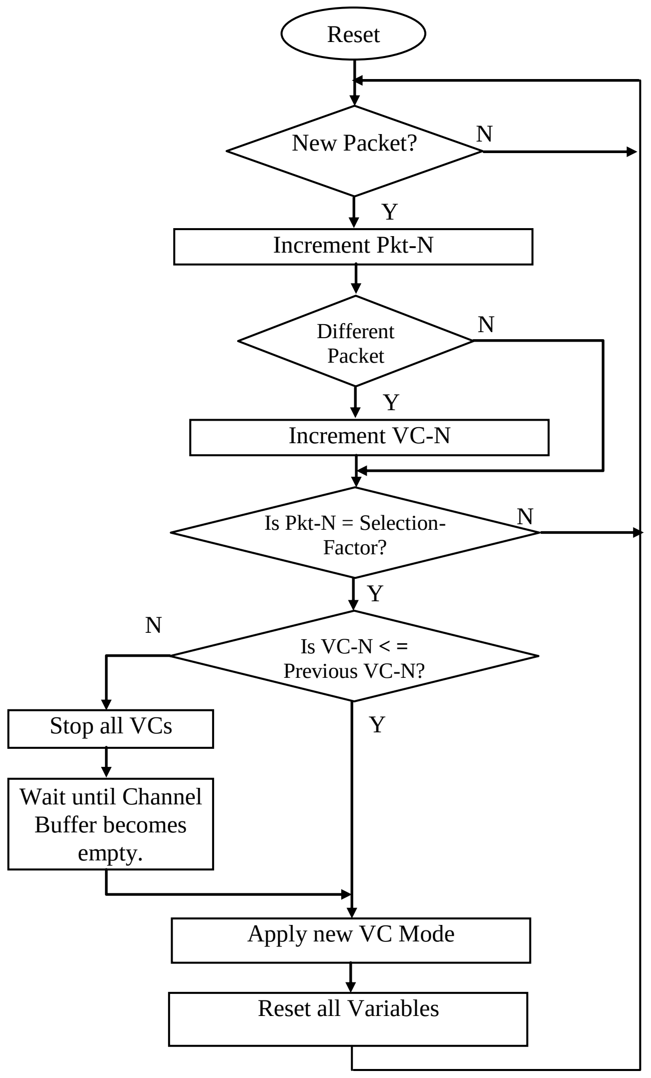
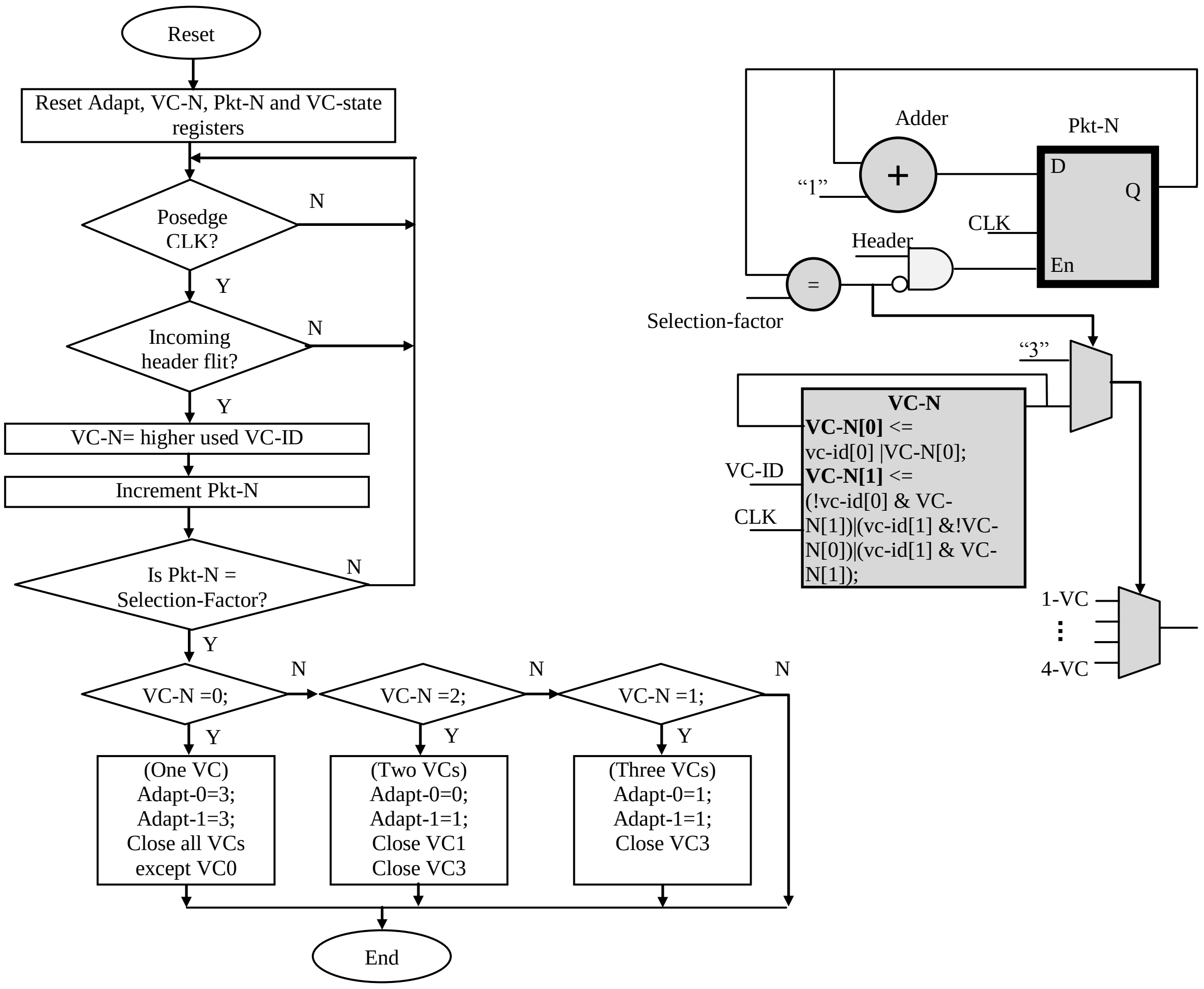


Figure 3.12: Flowchart Process of the Dynamic Mode-Selector

3.4.2 Deterministic Mode Selector

As mentioned earlier, switching from a higher mode to a lower mode is simple. It has no negative impact on communication and incurs very little cost for an NoC system. These advantages have encouraged us to introduce a deterministic routing based NoC architecture. In deterministic routing, packets with similar routing information always pass through a specific route. Therefore, when an embedded application is mapped to NoC, the number of various packets passing through a channel will be constant throughout the communication. Figure 3.13 illustrates the process flow and schematic diagrams of an efficient and simple deterministic *Mode-Selector*. The process performs once for an input-port. In fact, after the number of incoming packets reaches a specific number (*Selection-Factor*), the mode is assigned based on the maximum number of VCs to be used. Assume that there are four VCs per input-port and the arbiter is designed deterministically and sequentially to assign new and varied packets to VC0, VC1, VC2 and VC3. In such a situation, the same packets will always enter the same VC.

On reset, all the registers of the *Mode-Selector* become zero that leads to 4VC mode as the default mode (i.e. a maximum number of VC). At the positive edge of the clock the *Mode-Selector* process determines whether an incoming flit is the first flit of the packet. If the *VC-ID* of a new packet is different than the *VC-ID* of previous packets, it will be saved in a register, *VC-N*. The number of packets is recorded in a counter, *Pkt-N*. When the number of new


Reset
Reset Adapt, VC-N, Pkt-N and VC-state registers
Posedge CLK?
N
Y
Incoming header flit?
N
Y
VC-N= higher used VC-ID
Increment Pkt-N
Is Pkt-N = Selection-Factor?
N
Y
VC-N =0;
N
VC-N =2;
N
VC-N =1;
N
Y
Y
Y
(One VC) Adapt-0=3; Adapt-1=3; Close all VCs except VC0
(Two VCs) Adapt-0=0; Adapt-1=1; Close VC1 Close VC3
(Three VCs) Adapt-0=1; Adapt-1=1; Close VC3
End
Adder
Pkt-N
"1"
+
D
Q
CLK
Header
En
=
Selection-factor
"3"
VC-N
VC-N[0] <= vc-id[0] |VC-N[0];
VC-N[1] <= (!vc-id[0] & VC-N[1])|(vc-id[1] &!VC-N[0])|(vc-id[1] & VC-N[1]);
VC-ID
CLK
1-VC
4-VC
Figure 3.13: Flow Process & Schematic of Deterministic Mode-Selector.

packets reaches to the *Selection-Factor*, the *Mode-Selector* process decides about the mode to be switched depending on the value of *VC-N*. In each selected mode, all the redundant VCs are closed. For example in the 2VC mode, VC2 and VC3 virtual channels will be closed. The deterministic *Mode-Selector* module can be designed by employing a much simpler logic. The modes of each channel can be assigned at the setup time by the user. This design requires each channel to have *k* registers to hold the modes signals (*adapt*), where *k* is the log_2 of the number of modes. A single-bit register is used for holding the *VC-close* signal, and one NOR gate is needed for the acknowledgement signal of each VC. We consider this as a simple scheme that clearly shows and evaluates the efficiency of our approach.

3.5 CVC and SMF Router Micro-Architecture

The micro-architectures of routers for CVC [50] and our proposed SMF mechanisms are presented in Figures 3.14 and 3.15 respectively. We assume same number of VCs and channel buffer capacity for both the architectures. The first advantage of SMF is the hardware saving in terms of multiplexer (MUX) and de-multiplexer (de-MUX) blocks. The size of input and output-

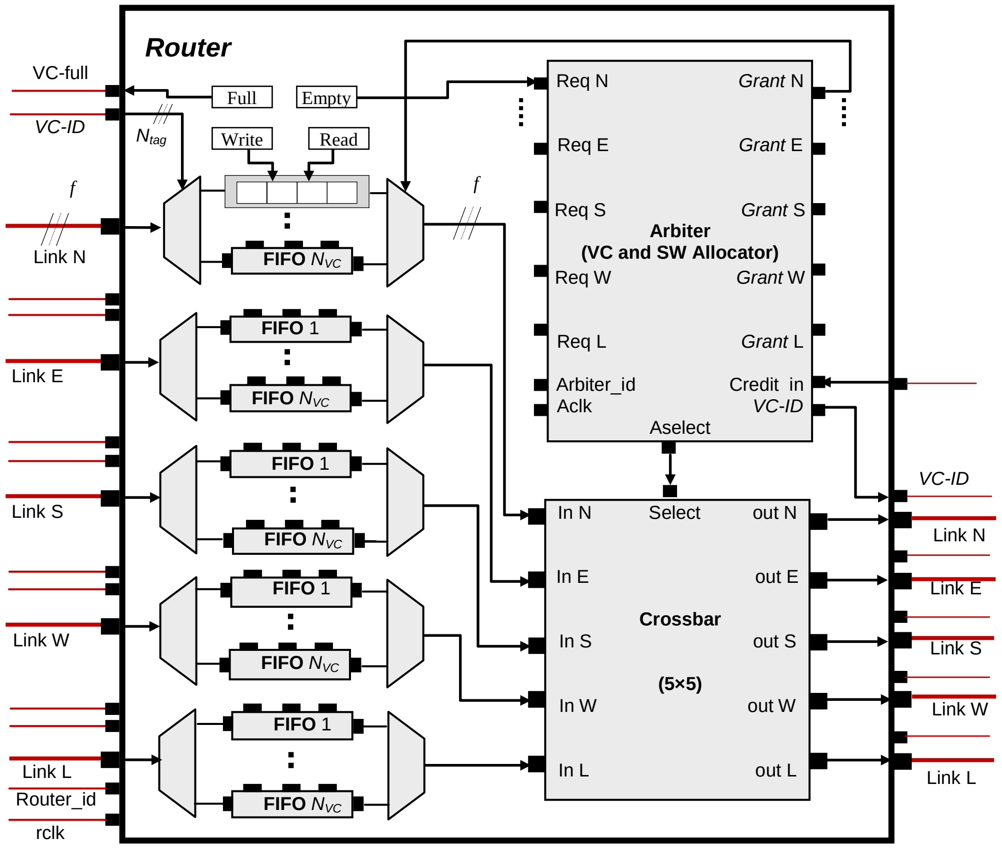


Figure 3.14: 5×5 CVC Wormhole Router.

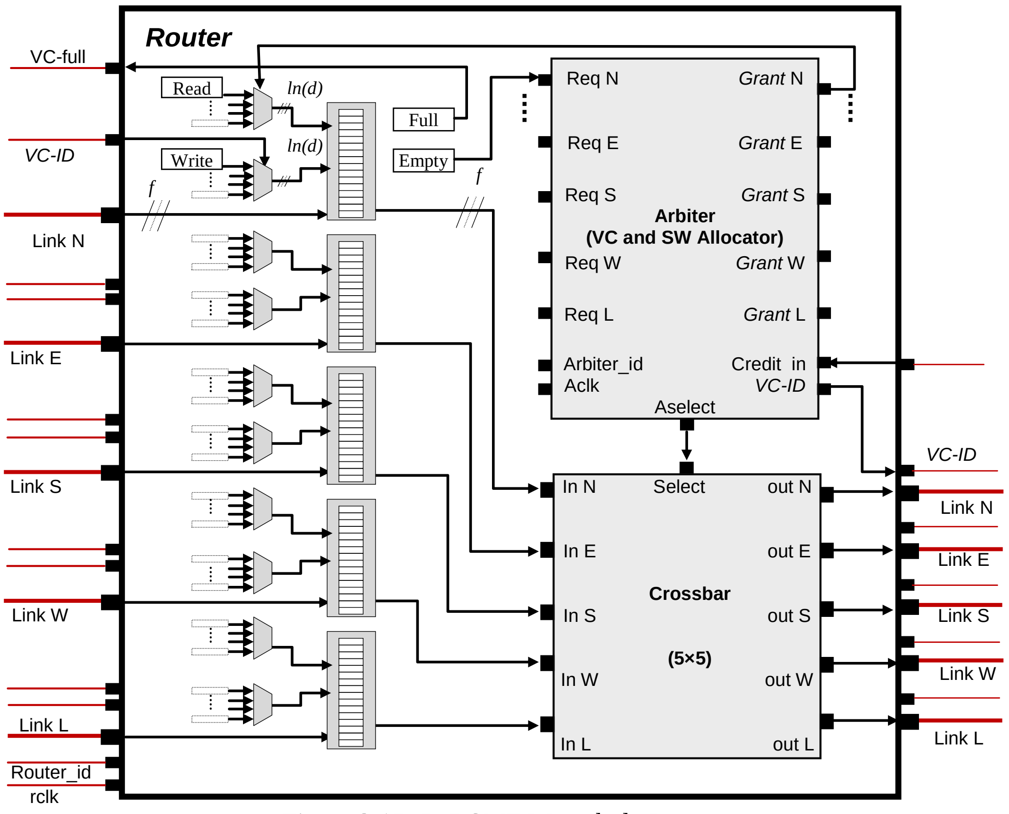


Figure 3.15: 5×5 SMF Wormhole Router.

ports of CVC MUX and de-MUX is the same as the flit size. However, the size of input and output ports of SMF multiplexers is equal to *log*$_2$(d), where d is the of VC depth. Figure 3.16 shows the schematic of a 5-to-1 MUX module. Assuming two transistors for an inverter gate, one needs (10+6) transistors to create the MUX module despite the fact that only 10 transistors participate in data communication. Consequently, the number of transistors to create a 5*fs*-to-*fs* multiplexer will be (10×*fs*)+6, where *fs* is the flit size.

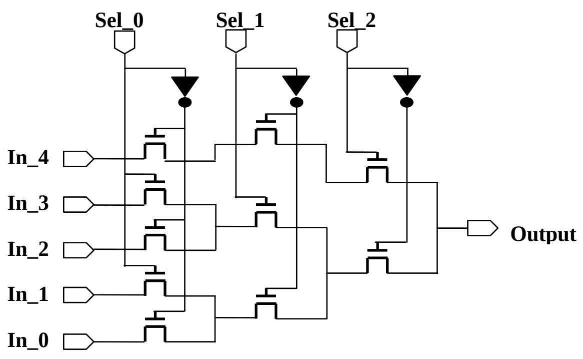


Figure 3.16: 5-to-1 Multiplexer Schematic.

Assuming the same formulation for a de-MUX, the percentage of saving transistors in SMF MUXs as compared to CVC can be determined by the following equation.

$\mathbf{R} = \{1- ([10\times 6 - log_2(d)]/[10\times fs+6])\}\times 100$ (1)

For a flit size of 32-bits and a VC depth (d) of 4 slots, the transistor saving **R** in SMF MUXs as determined by Eq. (1) will be almost 92%. In order to determine the power based advantage of SMF router model with the CVC models, we can use a comparative method as follows. One of the implementations of the crossbar module is based on MUXs.

Figure 3.17 illustrates the schematic of a MUX-based crossbar switch. There is a 5*fs*-to-*fs* MUX for each output-port in the crossbar. If we assume five VCs per channel in a CVC router, the size of MUX or de-MUX of each input channel will be 5*fs*-to-*fs*. Assuming the same hardware elements for MUXs in the input channels and the crossbar, the total hardware elements of MUXs and de-MUXs of input channels are two times of those in the crossbar switch. Furthermore, power consumption breakdown of a router in NoC based 80-tile Teraflops processor is shown in Figure 3.18, where a crossbar consumes 15% of the total power of Teraflops processor router [51]. We can assume the same power breakdown for our CVC router. Therefore, the MUXs and de-MUXs of a CVC router consume two times i.e. 30% of the total power of router. It can be determined that an SMF router consume 27.6% (92% × 30%) less power as compared to a CVC router in terms of MUXs used in the router by employing all the above assumptions.

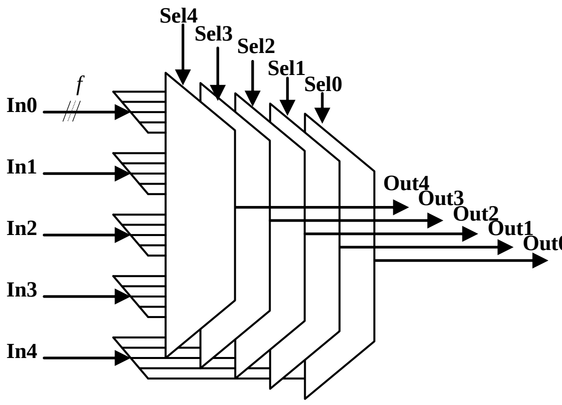


Figure 3.17: A Multiplexer-based Crossbar Switch

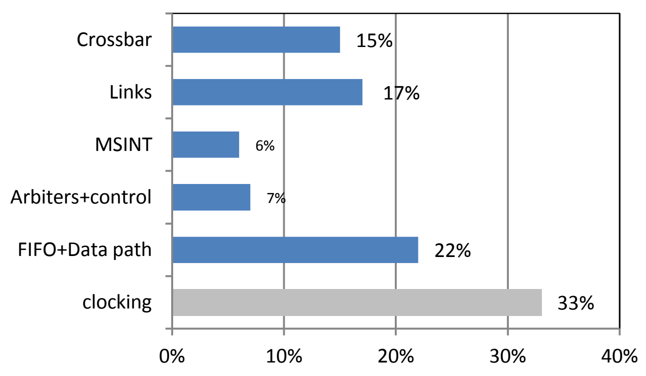
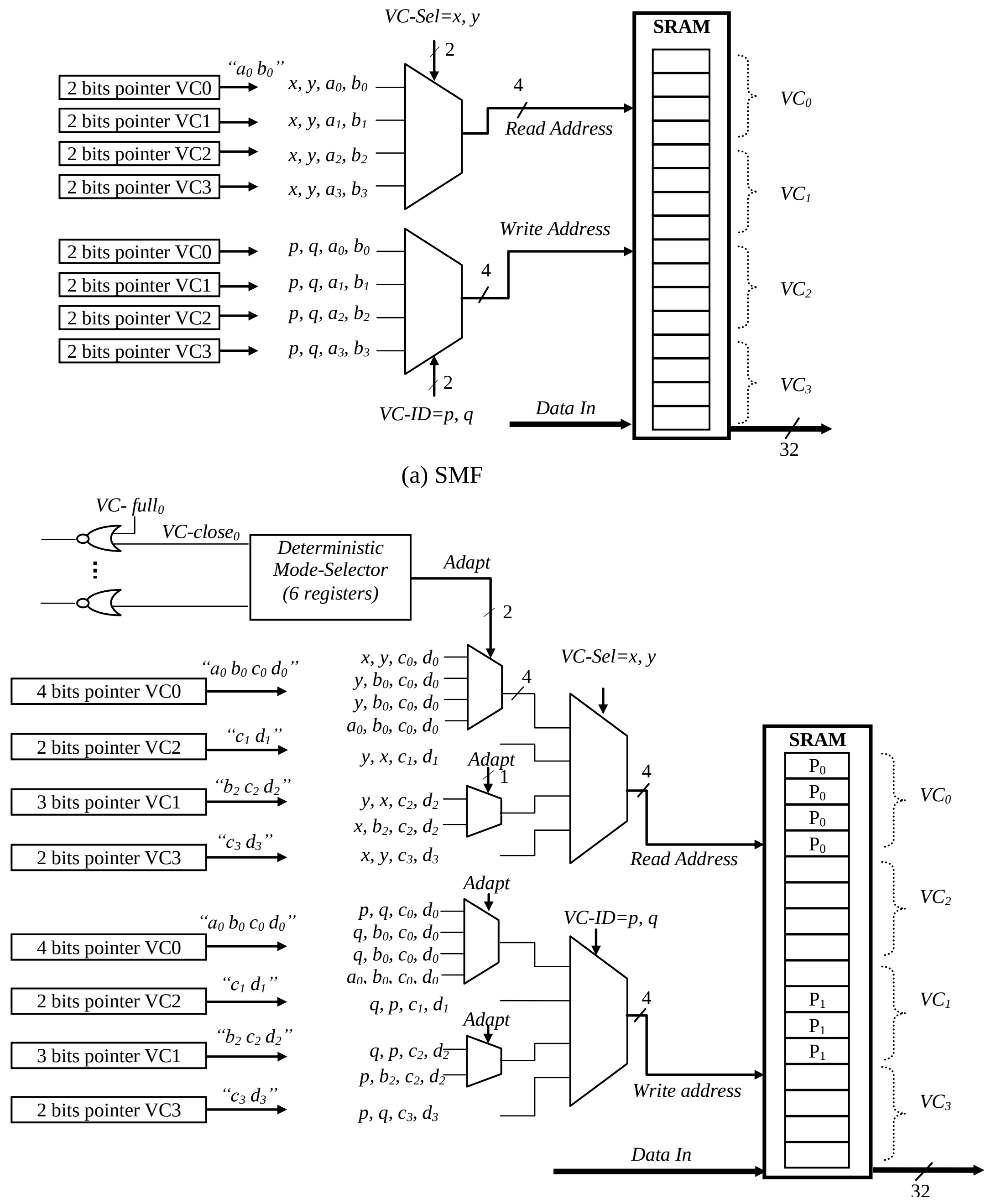


Figure 3.18: Power Breakdown for a CVC NoC Router

3.5.1 Adaptivity Hardware in SAMF Architecture

The detailed architectures of SMF and SAMF models are illustrated in Figure 3.19. However, the SAMF architecture is a trimmed version of the architecture presented earlier in Figure 3.11, where the mode selection is of deterministic type as shown in Figure 3.13. Both architectures are configured for a maximum of four VCs, where the depth of each VC is of four slots.

Each input of SMF MUX can be 2-bits because *x*, *y* and *p*, *q* signals can be directly connected to the read and write addresses of the SRAM respectively. Some extra hardware is needed in the SMF architecture to provide adaptivity feature as follows. VC0 takes part in all the four modes, which requires the pointers of four bits size. To involve VC1 in the 2VC mode, its place is exchanged with the VC2. The VC1 takes part in 4VC, 3VC and 2VC modes and it has the same size in 3VC and 2VC modes, and therefore the size of its pointers is three bits. Virtual channels, VC2 and VC3 take part only in the 4VC mode, and therefore the size of their pointers is of 2-bits. The sizes of *VC-Sel* and *VC-ID* multiplexers of SAMF are increased to four bits to support SRAM address for 1VC mode. Signals, c_0, d_0 and c_2, d_2 can be directly connected to the inputs of next multiplexers, and therefore two 2-bit wide 4-to-1 multiplexers and two 2-bit wide 2-to-1 multiplexers are required to switch the *Read* and *Write Pointers* to implement all the four adaptivity modes.



(a) SMF

(b) SAMF

Figure 3.19: SMF and SAMF Architectures

3.5.2 Synthesis of SAMF Router

To analyze the area and power overhead, the NoC routers for VC-free (VC free), CVC [49, 50], SMF and SAMF architectures are implemented in structural Register Transfer Level (RTL) Verilog and then synthesized using the Synopsys Design Compiler for 32nm Generic Library and Altera FPGA (Cyclone IV). The resulting designs operate at a power supply of 0.85V and a clock

frequency of 100 MHz. Each router has five input-ports with 4 VCs per input-port, where each VC is of two-flit deep, and flit size is 128 bits. The crossbar has identical architecture for the four (VC-free, CVC, SMF and SAMF) routers, and also the same arbiter architecture (Arbiter 4-VC) performs arbitration for the three (CVC, SMF and SAMF) routers. The comparison of area and power overhead of four routers is shown in Table 3.2. It should be noted that all the four routers have equal buffer space of 8 slots per input-port. An important characteristic of high scaled CMOS technology like 32 nm is that the static (leakage) power supersedes the dynamic power.
For example, the average dynamic power of ports includes almost 4% of their total powers as shown in Table 3.2. This characteristic indicate that the power is more or less proportional to the hardware overhead of a design than its functionality. In other words, the more cells consume more static power and the synthesis results given in Table 3.2 also confirm it. As mentioned in the previous section, the CVC input-port incurs an overhead due to bigger MUX and de-MUX blocks. This overhead leads to a bigger CVC input-port than our proposed SMF or SAMF input-ports in terms of area and power consumption.

Table 3.2. Synthesis Results for 32nm Technology and FPGA

Component	**ASIC Design (32 nm Generic Library) saed32rvt_ff0p85v125c**		**FPGA Design (Cyclone IV) EP4CE115F29I8L**	
	Total Area (*μm^2*)	Total Power (*μW*)	Combin. logic elements	Registers *(bits)*
Ports VC-free	78590	1570 (65*)	3295	5795
Ports CVC	88840	1820 (80*)	6550	5820
Ports SMF	80440	1595 (55*)	3485	5840
Ports SAMF	82870	1715 (105*)	3780	5910
Arbiter VC-free	2023	111 (43*)	330	60
Arbiter 4-VC	14274	644 (202*)	2682	280
Cross-bar	6499	121 (6*)	256	0
Router VC-free	87112	1802 (114*)	3881	5855
Router CVC	109613	2585 (288*)	9488	6100
Router SMF	101213	2360 (263*)	6423	6120
Router SAMF	103643	2480 (313*)	6718	6190
CVC/VC-free	20% extra	30% extra	59% extra	4% extra
SAMF/VC-free	16% extra	27% extra	42% extra	5% extra
SMF/CVC	8% saving	9% saving	38% saving	0.3% extra
SAMF/CVC	5% saving	4% saving	34% saving	1.5% extra
SAMF/SMF	2% extra	4.8% extra	5% extra	1% extra

* Dynamic Power

As illustrated in Table 3.2, the SAMF model provides area and power savings of around 5% and 4% as compared to those of CVC model respectively. We assumed that a router consists of an arbiter, a crossbar switch and 5 input-ports with equal-size input-port buffers for all the router designs including VC-free, CVC, SMF and SAMF. The synthesis results for Altera FPGA also confirm this trend and show the advantage of SAMF model as compared to CVC model in terms of combinational logic cells. The overhead related to the MUX and de-MUX blocks of CVC leads to smaller number of combinational logic cells for SAMF. In this way, an SAMF router saves 34% combinational cells at the expense of using 1.5% extra registers when compared to a CVC router. We have further experimented and evaluated the efficiency of our SAMF architecture and the results are presented in the next section.

In order to illustrate the approximate size of CVC [1, 49] and our SAMF, the area and power parameters for a basic VC-free NoC router is also synthesized and evaluated. Our SAMF based router consumes 16% additional area and 27% additional power as compared to a VC-free router. However, this extra hardware area and power is less than the area and power consumption for a CVC router model proposed by Dally and Towels [1, 49]. From the router architecture point of view, arbiter of a VC-free router is much simpler and consumes less power when compared with an SAMF or CVC arbiter. The main reason for SAMF or CVC routers having a larger area and power than a VC-free router is due to the complexity of their arbiter as confirmed by our synthesis data presented in Table 3.2. However, to achieve higher or comparable performance for a VC-free NoC, injection control and intelligent NoC level measures are needed that will also incur additional cost we could not include in the synthesis of a VC-free NoC router. The extra hardware associated with our SAMF VC mechanism provides a significant performance gain as compared to CVC and VC-free mechanisms. As indicated earlier, our SAMF router consumes less area and power as compared to CVC. Moreover, when compared to VC-free router, SAMF extra area and power is 16% and 27% respectively as compared to 20% and 30% extra area and power consumed by a CVC router.

3.6 Experimental Results

3.6.1 Adaptivity of SAMF Mechanism

We have evaluated the adaptivity level of our SAMF architectural model as compared to that of CVC and Dynamically Allocated Multi Queues (DAMQ) [14] based models. As discussed earlier in Section 3.3, the CVC mechanism is similar to our SMF approach in terms of its functionality. Therefore the performance evaluation presented here is also acceptable for the SMF mechanism. We demonstrate the efficiency of SAMF mechanism by two experiments. In the first experiment, we evaluate the SAMF and CVC models for two NoC applications. We mapped MPEG4 Decoder and AV Benchmark [52] to 3×4 and 4×4 mesh homogeneous NoCs. Then we investigated the effect of adaptivity modes in terms of chip area, power and performance of NoCs.

In the second experiment, three models: CVC [1, 49], SAMF and DAMQ [14] are evaluated. A 4×4 mesh NoC with fixed traffic and high contention is configured for better evaluation of the SAMF model. The DAMQ model is based on the Link-List design presented by Evripidou *et al*. [14]. We have already described our implementing architecture of Link-List DAMQ (LLD) design in Section 2.4. First of all, we explain the timing drawback existed during flit arrival and departure of LLD mechanisms. As mentioned before, the LLD design is a table-based mechanism where a central table that contains registers directs the data flow mechanism. The registers are updated at one edge of the clock cycle. This property of registers causes the table-based mechanism such as LLD [14] or ViChaR [13] to take one clock cycle longer than a static mechanism such as SAMF or CVC that do not employ tables.

To clarify this claim, we provide a brief example of the flit arriving/departing process in the two mentioned mechanisms. In the table-based DAMQ mechanism, the arriving/departing of a flit can be described as follows. (1) The flit arrives the router (e.g. at the +ve edge). (2) The credit of flit leads the flit to be stored in the input-port buffer and the flit information to be recorded in the input-port table (e.g. at the -ve edge). (3) After arbitration the flit cannot go out at the following clock edge because its information should be recorded in the table, so it can go out one clock cycle after it is recorded (e.g. at the -ve edge). (4) If we assume all the routers have the same timing process, the credit of the flit should reach at the same edge in the downstream router i.e. at the following two clock edges (e.g. at the -ve edge). For the above example, the arriving/departing of a flit takes two cycles (4 clock events). In our proposed mechanism, the arriving/departing of a flit take one cycle as follows. (1) A flit arrives at the SAMF router (e.g. at the +ve edge). (2) The credit of the flit leads the flit to be stored in the input buffer (e.g. at the -ve edge). (3) After arbitration the flit can exit the router at the following clock edge (e.g. at the +ve edge). (4) The credit of the flit goes out at the following clock edge (e.g. at the -ve edge). We consider the above timing process in our simulation results in this way that the arriving/departing of a flit in the DAMQ (Link-List based) takes 2 clock cycles and one clock cycle in the SAMF model. It should be considered that the deterministic switching mode in SAMF mechanism takes a small time interval of the communication (e.g. a maximum of 16 clock cycles to evacuate a 16-slot input-port buffer). Therefore, this tiny time is over-compensated by a larger performance improvement of our SAMF approach. FAANOS is our in-house SystemC based NoC simulator [53], which is used to measure the two important metrics of NoCs such as throughput and latency. For hardware chip area and power, we used Synopsys Design Compiler with 32nm Generic Library with the same technology configurations employed earlier in Section 3.5.2. The resulting NoC hardware operates at 100MHz and 0.85V power supply.

3.6.2 Experiment Setup

We setup our FAANOS simulator for an SAMF NoC and then apply the adaptivity modes of the SAMF architecture. The topology used for NoC is mesh and the communication of packets follows a deterministic XY routing algorithm. Due to the deterministic routing, the SAMF mechanism can utilize a deterministic *Mode-Selector* (see Figure 3.13) that applies the new VC configuration after 16 packets received by each input-port. The packet communication is in the form of parallel wormhole switching where the flit size is equal to the channel width. Each flit is of 128-bits and a packet can have ten flits. Assume that the time delays of channel links are negligible as compared to the time delay of a router. As mentioned earlier, we evaluate the adaptivity modes in two parts. In the first part of experiment, two application specific NoCs illustrated in Figures 3.20 and 3.21 are evaluated.

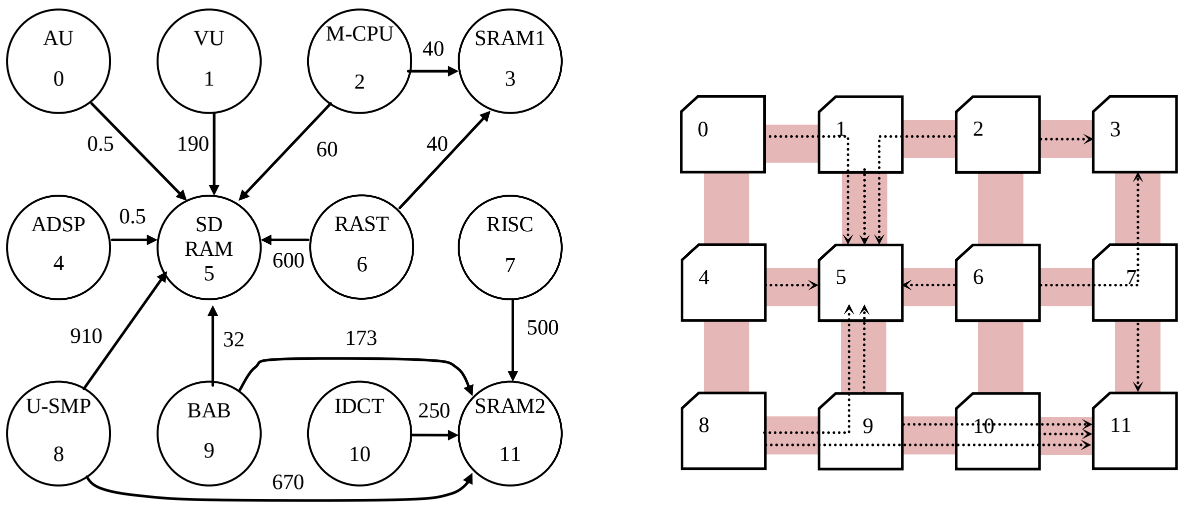


Figure 3.20: MPEG4 mapping core graph to a Mesh Topology and its XY routing.

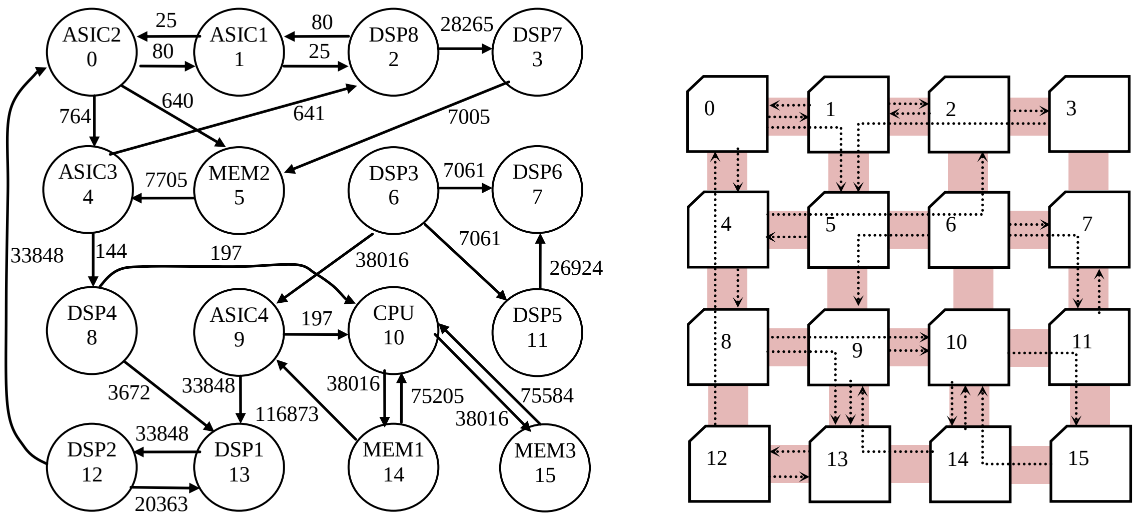


Figure 3.21: AV Benchmark mapping core graph to a Mesh Topology and its XY routing.

It is assumed that the size of buffers is constant in every physical channel. In the adaptivity mode, the channel buffers are divided to multiple VCs statically. The number of VCs per channel is determined according to the number of packets that can simultaneously pass through the physical channel. For example, if only two different packets pass through a physical channel at the same time, the channel is divided into two VCs. We also evaluate the SAMF model as compared to a VC-free model with the same input-port buffer size. In the second part of the experiment, we employ a 4×4 mesh NoC topology given in Figure 3.22. All the source cores send packets randomly and uniformly to one destination core at a time. The source cores are clocked at 40*ns*, and the destination and router modules are clocked at 10*ns*. As we argued earlier in section 4, the SAMF model improves the buffer utilization by increasing the buffer depths of VCs. An increase in the VC depth reduces contention and, eventually increases performance. We deliberately increase the contention in the NoCs to investigate the adaptivity of our SAMF model properly. For this reason, one destination is chosen for all the source cores to create high traffic around the destination core that creates more contention. Figure 3.22 shows this condition for destination core #10. In XY routing if all the source cores send packets to destination #10 simultaneously, the north side input channel of destination will have eight flits request.

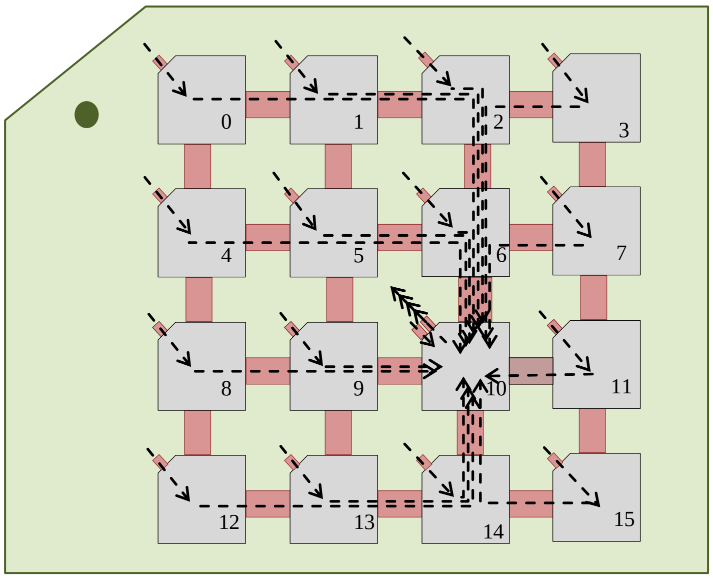


Figure 3.22: Fixed Communication for a 4×4 Mesh NoC

3.6.3 Experimental Results and Analysis

The routing graphs show mapping of two applications (MPEG4 Decoder and AV Benchmark) of a Mesh NoC as shown in Figures 3.20 and 3.21. The XY routing (arrow lines) specifies the number of VCs needed for each channel. For example, three arrows toward the north input channel of router #5 (Figure 3.20) means three packets will pass through the channel. Three packets require three VCs to service the three packets without any contention. In the CVC model, all the channels have the same number of VCs. Some channels may have some idle VCs and their buffers can be used by the other busy VCs in our SAMF model. For example, when a channel requires only one VC, all the buffer space reserved for the channel is used by that one VC.

The efficiency of SAMF model can be better evaluated under two configurations of NoC. First of all, the NoC should be configured in such a way that high contention traffic occurs in the communication. Under high contention, the bigger VC buffer size leads to more data to store in the channel buffer. Subsequently, congestion will diminish and contention will be reduced in the NoC. Secondly, the NoC configuration should be in such a way that with the increase in VC buffer size, the performance will improve. Our simulator creates two configurations as follows. For the first condition, we create a configuration for the NoCs to deal with contention. In the case of MPEG4 decoder, 2-3 VCs can create contention in NoC, and in the AV application, only two VCs can create contention. These conditions have helped us to investigate the SAMF model for the MPEG4 and AV benchmark NoCs as follows. The simulation results given in Figure 3.23 shows that the average throughputs for MPEG4 and AV application NoCs are increased by 5% and 2% respectively for SAMF as compared to those of CVC.

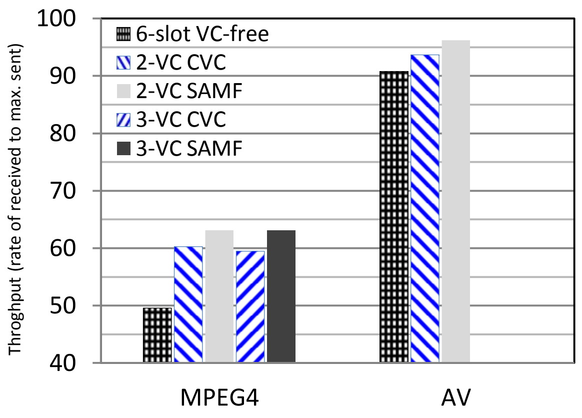


Figure 3.23: Throughput for Different NoC Applications

Synopsys Design Compiler generated router hardware and power parameters are presented in Figure 3.24. These results are based on 32nm Generic Library for 0.85V power supply and an operating frequency of 100MHz. These results demonstrate that the average router power consumptions in MPEG4 and AV NoCs are decreased 4% for SAMF as compared to those of CVC.

The average router areas for MPEG4 and AV NoCs are decreased 6% in the SAMF as compared to a CVC as demonstrated in Figure 3.25. Figures 3.23 to 3.25 also show that the SAMF model improves the average throughput by 27% as compared to VC-free model in exchange to the extra cost of 11% in area and 18% in power consumption in MPEG4 application

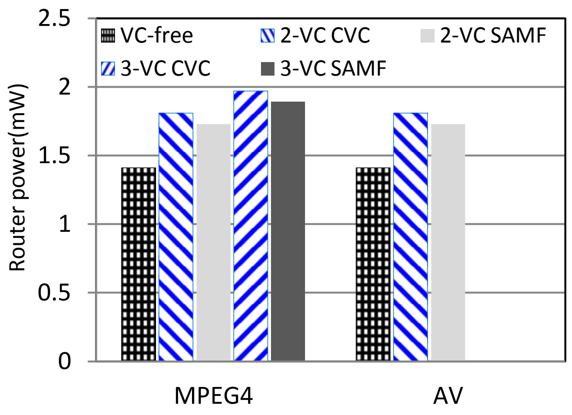


Figure 3.24: Power of a Router for Different NoC

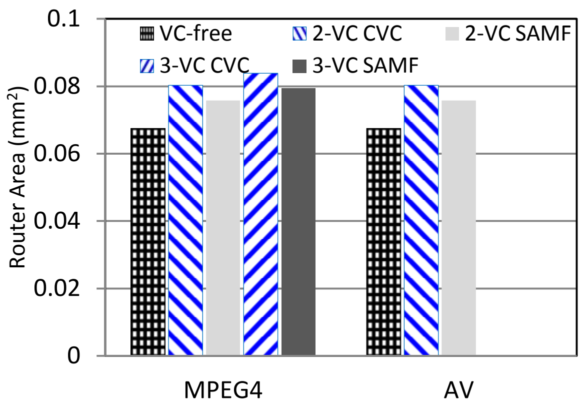


Figure 3.25: Area of a Router for Different NoC

and with the same size of input-port buffer of 6. The performance results indicate a little improvement for SAMF model for applications such as MPEG4 decoder and AV benchmark.

In the second experiment, the CVC, SAMF and DAMQ models are evaluated. The size of the input-port buffers in CVC and SAMF varies at each VC number. In other words, each input-port buffer in the S-slot SAMF or CVC configurations has the size of VC number multiply by S. For example, for 8 VCs and 2-slot SAMF configuration, the size of the input-port buffer is 16.

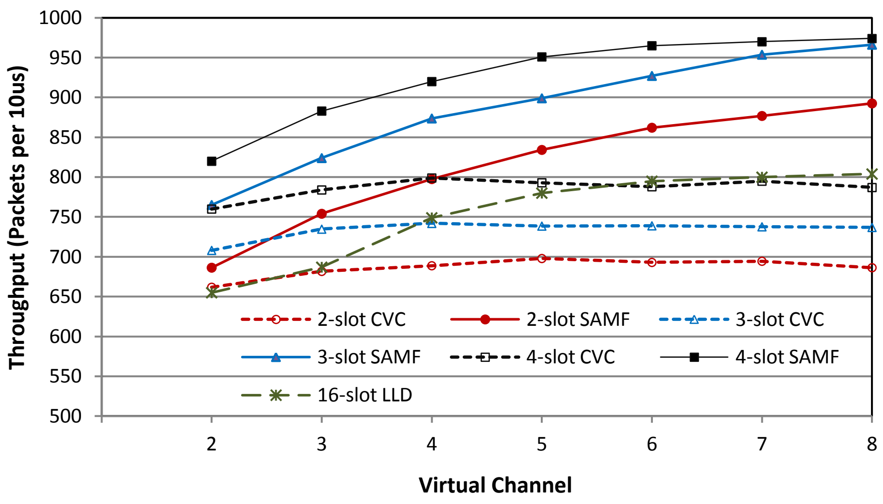


Figure 3.26: Throughput for High-Contention NoC Traffic

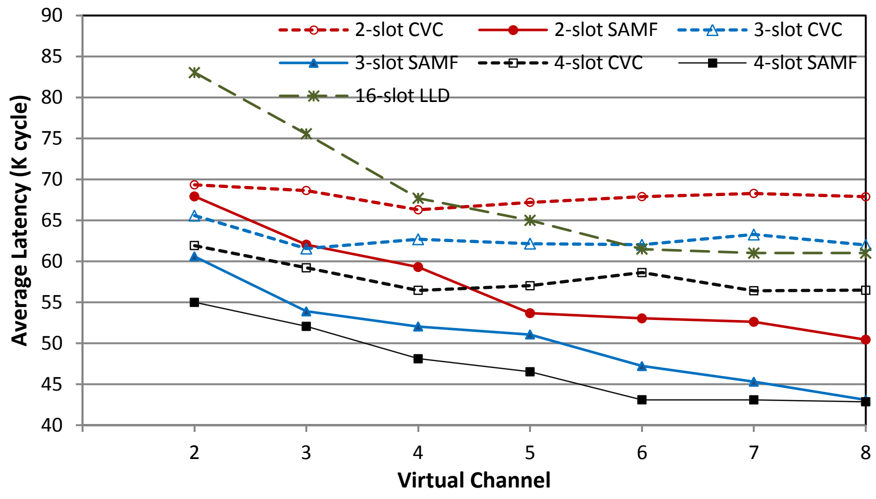


Figure 3.27: Average Latency for High-Contention NoC Traffic

However, the size of DAMQ input-port buffer is constant and equal to 16 slots (the slot size is equal to the flit size) in all the recorded results. A high efficiency of the adaptivity of our SAMF model can be noticed under high contention traffic in the NoCs as shown in Figure 3.22. All the source cores send packets to one destination (sink) core (#10 in this setup). The simulation results for NoC throughput are shown in Figure 3.26. These results confirm that the throughput increases when the number and depth of VCs are increased. This is a much improved situation for the SAMF model. The average throughputs of SAMF are 19% higher than those of the CVC. This improvement can also be seen in the latency graphs shown in Figure 3.27. The average latencies of CVC models are 23% higher than those of the SAMF model.

The hardware requirement results from the synthesis also show the advantage of SAMF model as illustrated in Figures 3.28 and 3.29. As mentioned before, this hardware advantage is due to saving in smaller multiplexers of SAMF input-ports. The average power consumptions of SAMF router is 4% lower than that of a CVC router, where the average area occupied by an SAMF router is 5% smaller than that of a CVC router. As mentioned earlier, for 32nm technology the static power dominates the dynamic power and one should expect the same trend shown in Figures 3.28 and 3.29. As illustrated in Figures 3.26 and 3.27, with four VCs, the performance of CVC models is almost constant and any increase in the number of VCs does not affect the throughput and latency of NoC. It is due to the Mesh topology of NoC where each router has five input-ports and for each output-port (in XY routing) four packet requests can occur simultaneously. Therefore, in the case of CVC, four VCs of a channel usually take part in the communication. Increasing the number of VCs (from four) will not improve performance. However, in the SAMF model, the buffers of these useless VCs are used for the other VCs and will improve the performance of NoCs. An interesting feature of the SAMF model is to extract higher performance when the number of VCs is less than the CVC model. For example, all the configurations of 3-slot SAMF NoCs are more efficient (higher throughput and lower latency)

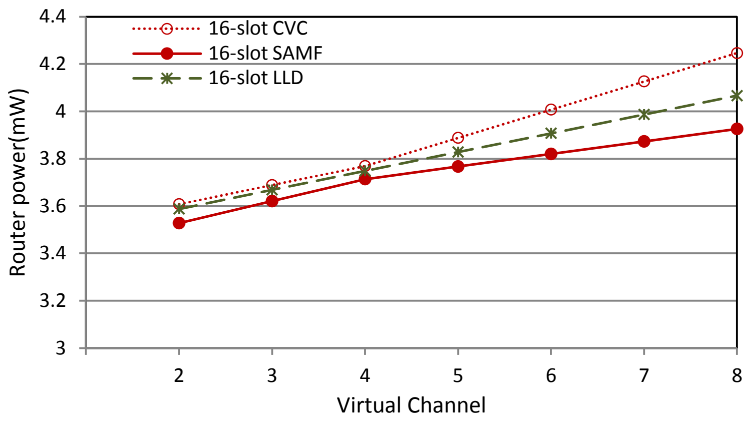


Figure 3.28: Router Power for High-Contention NoC Traffic

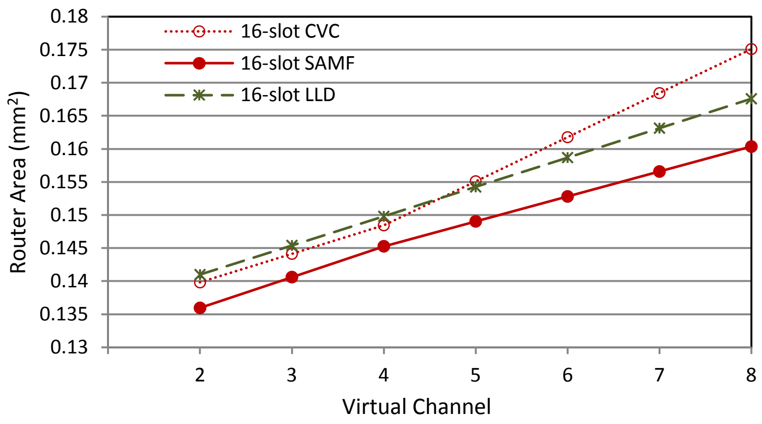


Figure 3.29: Router Area for High-Contention NoC Traffic

than all the configurations of CVC based NoCs. This feature becomes more interesting when the power and chip area are also considered.

Another model simulated in the second experiment is the LLD model. As we already discussed, the LLD model has two major drawbacks. The first drawback is the longer flits arrival/departure (one extra cycle) as compared to that of SAMF or CVC models. The second drawback of LLD mechanism is the monopoly of input-port buffer by a VC especially in high contention traffic. This condition may cause a deadlock in communication [15]. Therefore, we expect lower performance of LLD as compared to that of SAMF as one can confirm from the results presented in Figures 3.26 and 3.27. The LLD model latency and throughput are 23% higher and 17% lower than those of 16-slots SAMF respectively. This should be considered that the input-port buffer size (2-slot, 8-VC) and (4-slot, 4-VC) for SAMF configuration is equal to 16. For the sake of fair comparison, the hardware requirement results for all the three models have the same port buffer size (i.e. 16-slot) in their routers. A LLD router is 5% larger than an SAMF router due to extra registers of the link-list table used to implement and to manage a LLD mechanism. The power consumption of LLD is mere 2% higher than that of SAMF model. The power overhead is due to the higher static power associated with a bigger LLD router as compared to SAMF model router.

3.7 Novelty of Approach

The approach (SAMF) presented in this chapter is a static adaptive VC organization that improves the buffer utilization and eventually the performance of NoC. It has following novelties.

- The SAMF data flow mechanism is as simple as that of a static VC organization, so it can benefit of fastest arrival and departure of data.
- The SAMF organization can easily switch to different static VC configurations according to the data flow traffic to provide the optimum buffer utilization and, subsequently higher performance.
- The extra hardware utilized for the adaptivity of mechanism is miniature as compared to the hardware of organization.

3.8 Summary

The virtual channel (VC) flow control mechanism is critical to the performance and energy problem of NoC. This mechanism suffers from some drawbacks such as contention, lack of high buffer utilization, HoL blocking and higher cost. First of all, we introduced a low cost VC mechanism (SMF) that has same functionality performance as a conventional static VC mechanism (CVC). An SMF router consumes 8% less area and 9% less power as compared to a conventional router that is confirmed by our synthesis results. We took the advantage of SMF architecture and introduced an adaptive model of VC (SAMF) to improve buffer utilization of an NoC channel. In a 4×4 mesh SAMF NoC with contention oriented traffic, 19% throughput and 23% latency improvements are gained with a 5% decrease in the area and 4% decrease in the power as compared to CVC. Another interesting feature of the SAMF model is to achieve higher performance for a lower hardware overhead as compared to the LLD (Link-List based DAMQ) model.

Chapter 4

Efficient Dynamic Virtual Channel Organization

In this Chapter, we present a state of the art micro-architecture of input-ports for dynamic VC organization. As mentioned in Chapter 1, most of the DAMQ organizations are table based, and they suffer from higher hardware overhead. Our DAMQ approach presented in this chapter does not use table, and it utilizes a circuit based mechanism for VC-based buffer organization. The description of our contribution is organized as follows. An overview of our approach is presented in Section 4.1. The micro-architecture called Efficient Dynamic Virtual Channel (EDVC) is discussed in detail in Section 4.2. The buffer access in our approach is improved by proposing two fast read and write pointers in Section 4.3. The novelty of the approach is described in Section 4.4. We compare the hardware requirement and the performance of EDVC with those of two DAMQ based VC organizations in Section 4.5. Finally, the concluding remarks are made in Section 4.6.

4.1 Overview

The simplicity of our EDVC mechanism is shown in the block diagram illustrated in Figure 4.1. A small table, *Slot-State* (Boolean value) is required to manage the EDVC mechanism. The depth of the *Slot-State* table is equal to the depth of input-port buffer.

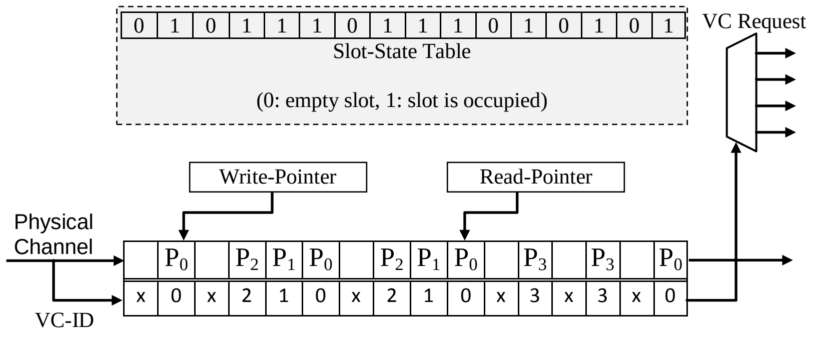


Figure 4.1: EDVC Input-Port Block Diagram

The EDVC mechanism utilizes the common features of DAMQ input-port to create a dynamic flow control. For example, The VC identification (*VC-ID*) is saved with the flit-data in the input-port buffer to assist in our efficient VC mechanism for issuing the *request* signals to the arbiter. *VC-full* and *VC-block* signals assist the VC organization to maintain the order of flits associated with each VC.

4.1.1 Simpler Communication in EDVC

In this section, we explain the EDVC mechanism when there is no contention in the NoC communication. In such a communication, the EDVC buffer works like a static VC (a parallel FIFO). Assume there is no blockage, each incoming flit and its *VC-ID* is stored in the buffer location pointed by the *write-pointer.* The flit pointed by the *read-pointer* is read and based on its *VC-ID* arbitrated and sent out of the buffer. Therefore, we expect that the flit arrival/departure time in EDVC is the same as that of a static VC organization and consumes three steps as shown in Figure 4.2. We employ asynchronous communication in our EDVC organization for NoCs. Following functions describe the working of EDVC in detail.

- Flit Arrival (Clk-edge #2): A *Credit-in* signal causes the incoming flit and its *VC-ID* to be saved in a slot pointed by the *write-pointer*. Meanwhile, the corresponding bit of the *Slot-State* table is set.

- Request Signal (Clk-edge #2): When the *read-pointer* points to a slot and its *Slot-State* bit is set (the slot containing data), a *Request* signal is issued according to the *VC-ID*. The arbiter will read the flit information and perform arbitration.
- Grant Signal (Clk-edge #3): If the requested output-port is open, the arbiter allocates the proper address for the crossbar switch and *VC-ID* before issuing a *Grant* signal to the input-port.
- Flit Departure (Clk-edge #3): The *Grant* signal causes the flit to leave the buffer. Meanwhile, the corresponding bit of the *Slot-State* table is reset.
- Credit Signal (Clk-edge #4): The high level of *Grant* at the negative clock edge causes the *Credit-out* and the *Grant* signals to be set and reset respectively.

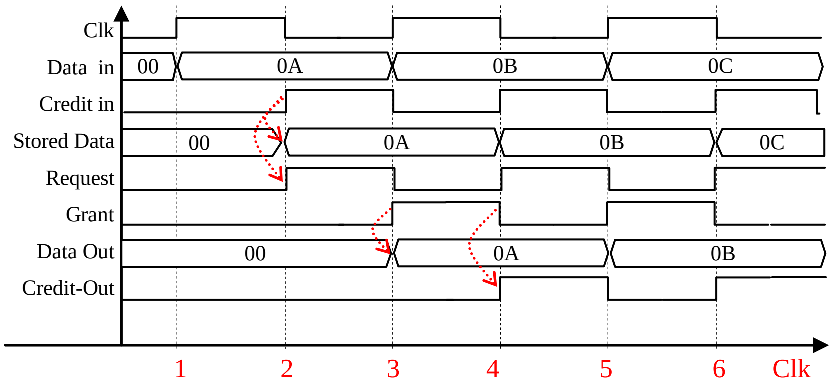
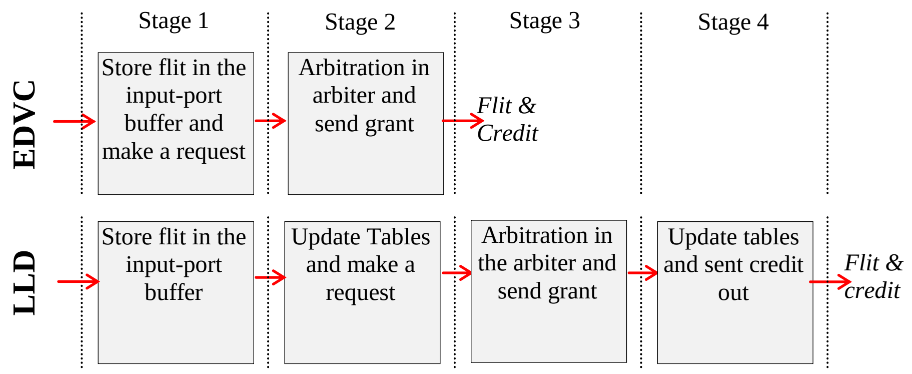


Figure 4.2: Three Steps of EDVC VC Flow Control.

In Figure 4.3, the above EDVC working process is compared with the LLD working process illustrated in Figure 2.13, as one can notice the LLD takes additional one clock cycle. If the requested output-port of a VC is closed (i.e. blocked), the arbiter issues the *VC-block*


Stage 1
Stage 2
Stage 3
Stage 4
EDVC
Store flit in the input-port buffer and make a request
Arbitration in arbiter and send grant
Flit & Credit
LLD
Store flit in the input-port buffer
Update Tables and make a request
Arbitration in the arbiter and send grant
Update tables and sent credit out
Flit & credit
Figure 4.3: EDVC vs. LLD Buffer Pipelines

signal to close the corresponding VC. Closing a VC means that a *request* is not issued and no flit enters the buffer for the VC. We further discuss the blocking condition later in this chapter.

4.2 EDVC Router Micro-Architecture

The structure of an NoC router with our EDVC input-port is shown in Figure 4.4. Our proposed EDVC router for a mesh NoC consists of five input-port modules, an arbiter and a crossbar switch as illustrated in Figure 4.4a.

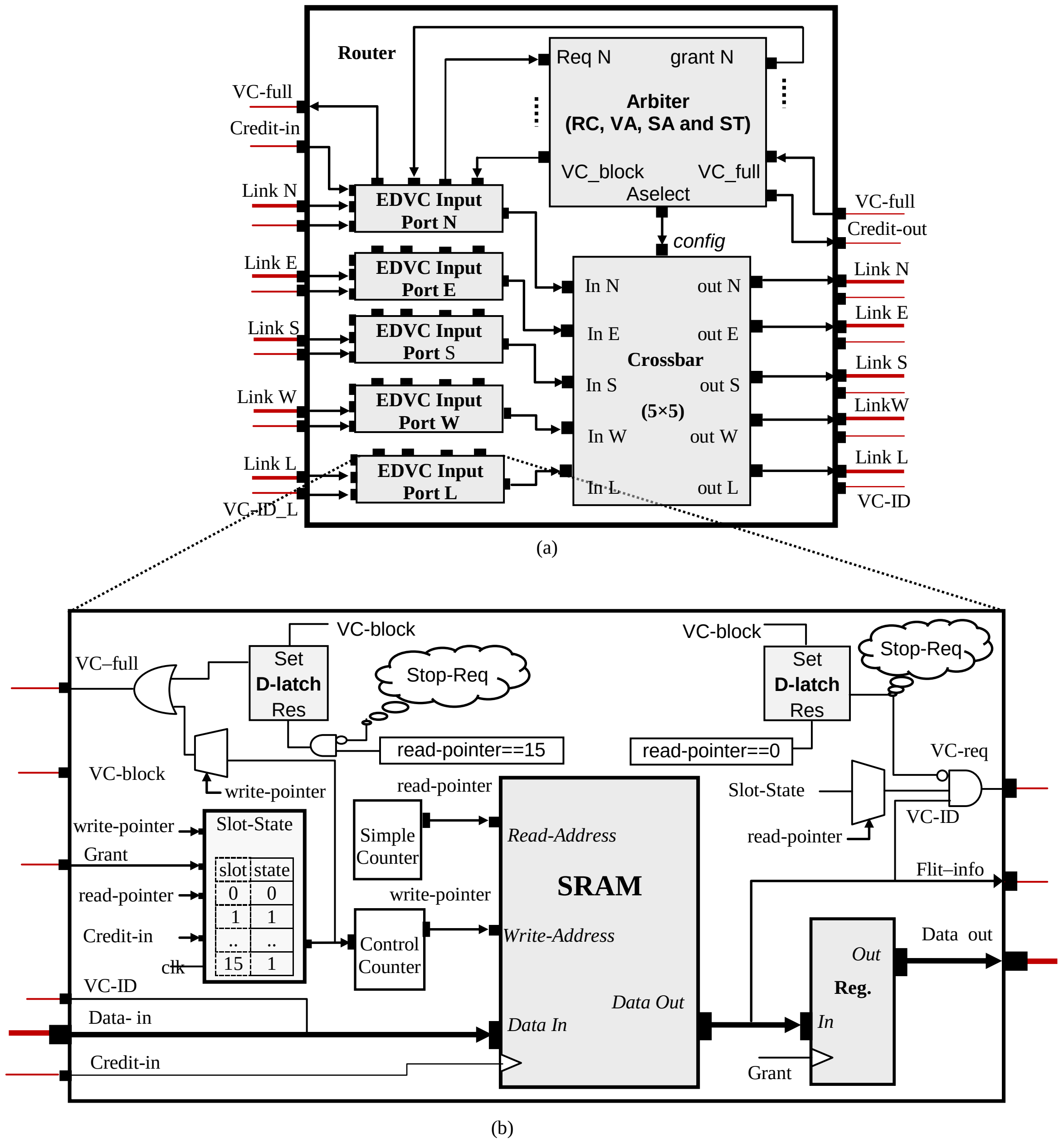


Figure 4.4: 5×5 EDVC Router and Input-Port Micro-Architecture.

In fact, it is similar to the LLD router architecture we presented earlier in Figure 2.11. However, the architecture of EDVC input-port shown in Figure 4.4b is much simpler in terms of less and efficient hardware and buffering hardware. The EDVC input-port micro-architecture includes an SRAM, a *Slot-State* table, two counters, and some other logic circuits and ports that are shown in Figure 4.4b. The SRAM module served as the input-port central buffer. The slot size of the SRAM is equal to the flit size plus *VC-ID* size (see Figure 4.1). The flit-data pointed by the *read-pointer* appears at the SRAM output. When the *Credit-in* signal is activated, the flit-data is stored in the SRAM slot pointed by the *write-pointer*. Various modules of the EDVC input-port are described in detail in the following sub-sections.

4.2.1 Slot-State Table

The process of the *Slot-State* table organization is the same as that of the LLD input-port illustrated earlier in the flowchart of Figure 2.14a. However, in the case of EDVC, the content of the *Slot-State* table is used to control the *write-pointer* and to issue the virtual channel request, *VC-req* signals to the arbiter.

4.2.2 Blocking Circuit

When the requested output-port of a VC is closed, the arbiter issues the *VC-block* signal for that VC. The *VC-block* signal sets both D-latches as can be seen in the upper part of Figure 4.4b. One D-latch prevents the blocked VC to be requested while the other is involved to generate *VC-full* signal to block the incoming flit for the full/blocked VC. These two conditions keep the blocked packet in the buffer and maintain its order until the blocking is removed. Consider the example of a channel buffer in Figure 4.5 showing a packet, P1 that was blocked and now its output-port is open. To read P1 flits in order, its request should first be enabled. After all the flits of P1 are read the VC can now accept new flits. We will discuss the removal of VC blocking in the following section.

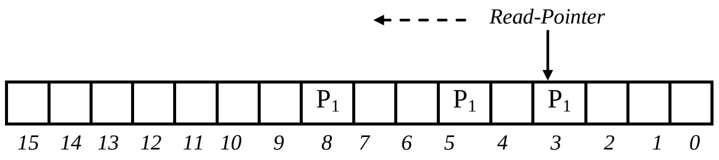


Figure 4.5: Packet P1 is blocked.

4.2.3 Enabling Blocked Request and VC-full

Consider the channel buffer example of Figure 4.5 again. The *read-pointer* increments from right to left in the buffer and the ordering of P1 flits is also from right to the left direction. Therefore, to remove the blockage of P1, its flit at buffer location 3 should first be read. To implement this mechanism, the blockage should be removed by starting the *read-pointer* from location 0. The circuits in the top-right corner of Figure 4.4b implement this mechanism such that a *VC-block* request is freed when *read-pointer* becomes zero.

The *write-pointer* can be anywhere when the blockage of packet P1 is removed. However, the writing of any new flit of P1 cannot be placed in-between the P1 flits residing in the buffer (e.g. locations 4, 6 and 7). To prevent such condition from occurring, the entire buffer should be read once to send out all the freed flits that were blocked. A typical logic circuit shown in the top-left corner of Figure 4.4b implements the mechanism. When the *read-pointer* reaches at the end of the buffer and the related request is free, the blocked *VC-full* is freed (reset).

4.2.4 Operation of Read and Write Pointers

The *read-pointer* works like a counter that counts the clock cycles as shown in Figure 4.6a. The *write-pointer* also works as a counter but it is controlled by the *Slot-State* table. It also counts the clock cycle when the slot of the input-port buffer is full as shown in Figure 4.6b. When the *write-pointer* points to an empty slot, it stops counting. When the data is stored in the slot, its corresponding bit is set and causes the *write-pointer* to increment at the next clock. If the following slot is already full, the input-port does not issue a *Credit* signal to upstream routers.

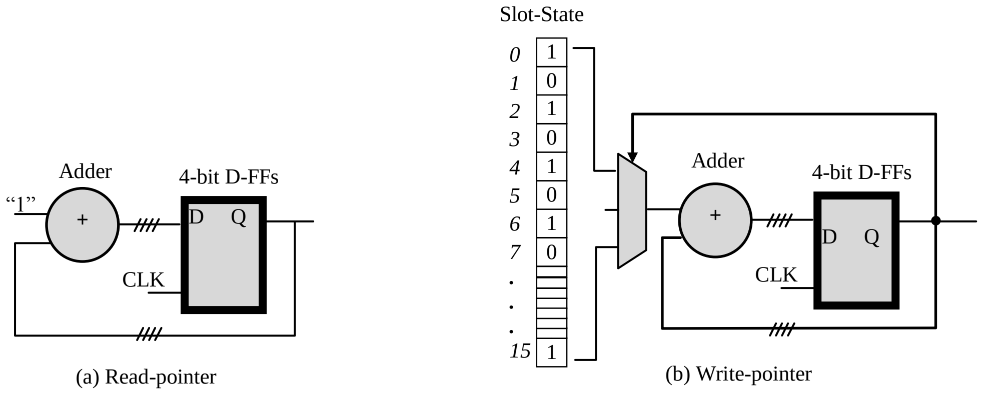


Figure 4.6: 4-bit Simple Read and Write Pointers

This will cause the upstream router to stop sending flits to the buffer. Moreover, when the slot is occupied, the *write-pointer* is incremented until it reaches an empty slot. When it reaches an empty slot, the *write-pointer* stops, and a *Credit* signal is simultaneously issued to the upstream router to resume sending flits. The *VC-full* signals implement the role of *Credit* signals. In other words, when the slot pointed by *write-pointer* is occupied, all the *VC-full* signals become set.

4.2.5 EDVC Closing and Requesting Approach

Table 4.1 lists the conditions associated with the closing and requesting operations of a VC (blocking circuit). The closing and requesting operations are actuated by the *VC-full* and *VC-req* signals respectively. There are three conditions according to the state of *VC-block*. The *VC-block* is issued to input-port by the arbiter to inform about the state of associated VC in terms of arbitration service. In fact, if the VC cannot succeed to win a free VC of the downstream input-port router, it becomes set. For condition 1, the *VC-block* is reset, so that the closing (*VC-full*) and the requesting (*VC-req*) operate in normal condition (see table). For condition 2, the *VC-block* become set, so that the mechanism stops receiving flits and requesting to the arbiter for the VC. In this condition, the order of VC flits inside the port buffer is kept until the blockage is removed. In condition 3, the *VC-block* switches from one to zero i.e. the VC can succeed in the arbiter. In case that the *read-pointer* ≠0, the VC issues no request to the arbiter and receives no incoming flit. The *read-pointer* continues incrementing becomes zero. At this point, the VC can only issues request to the arbiter leading its flits to exit the router without new flits enter the VC. The *read-pointer* continues incrementing until reaches to the end of port buffer. At this point, the closing and requesting operations of the VC return to the normal condition.

Table 4.1. Closing and Requesting Operations of EDVC Mechanism Associated with a VC

condition	*VC-block*	*Stop-Req*	*VC-req*	*VC-full*
1	0	0	X: Normal 1: *read-pointer* points to a flit of the VC. 0: *read-pointer* points to no flit of the VC.	X: Normal 0: *write-pointer* points to empty slot. 1: *write-pointer* points to occupied slot.
2	0 to 1	1	0: No request	1: No incoming flit
3	1 to 0	1: *read-pointer* ≠ 0	0: No request	1: No incoming flit
		0: *read-pointer* = 0	X: Normal	1: No incoming flit
		0:*read-pointer* = 15	X: Normal	X: Normal

4.3 Improving Input-Port Buffer Access

The input-port mechanism presented earlier is very slow especially when the NoC is crowded with packets. This problem stems from the structures of the *read-pointer* and *write-pointer* that are implemented as simple counters. For example, the situation illustrated in Figure 4.7, shows the *read-pointer* in front of the *write-pointer* such that the packet P1 is read after 13 cycles after it was written. The same scenario occurs for the *write-pointer*. Figure 4.8 shows a situation where the write at location 15 occurs 14 cycles after a write at location 1. In fact, there is no write in the buffer during 14 cycles. These latencies amongst the buffer write and read events lead to performance deterioration. We present a faster buffer organization for fast read and write.

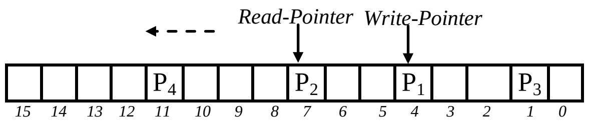
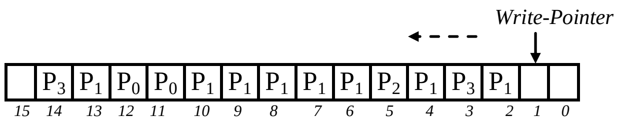


Figure 4.7: Fast Read and Write pointer



Write-Pointer
P_3 P_1 P_0 P_0 P_1 P_1 P_1 P_1 P_1 P_2 P_1 P_3 P_1
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Figure 4.8: Write at location 15 occurs 14 cycles after a write at location 1.

4.3.1 Fast Buffer Write

Ideally, after a write takes place, the *write-pointer* should point to the next empty slot anywhere in the buffer at the next clock edge. Considering the situation shown in Figure 4.8, it only happens when the *write-pointer* points to location 15 at the next clock cycle after writing at location 1. Figure 4.9 illustrates our proposed circuit for a 4-bit *write-pointer* that facilitates a much faster buffer write.

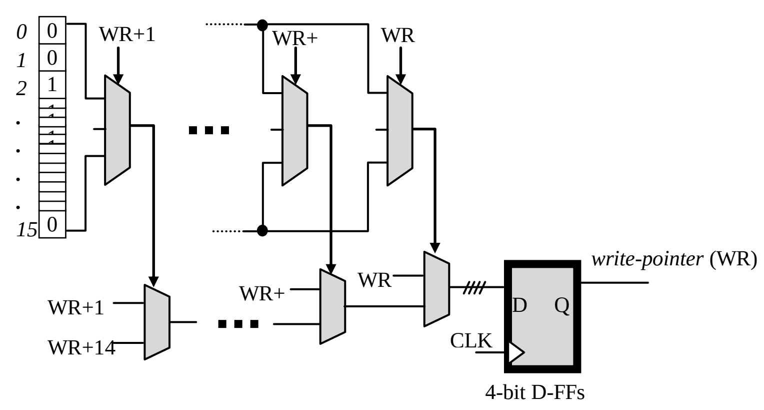


Figure 4.9: 4-bit EDVC Fast *Write-Pointer*

The *write-pointer*, WR points to the first empty slot of the channel buffer and remains there until a write occurs. After a buffer write, the corresponding *Slot-State* is set to 1. The *write-pointer* then points to the next empty buffer slot at the next clock edge.

4.3.2 Fast Buffer Read

The ideal condition that enables a faster buffer read to take place is when the *read-pointer* points to the next occupied location of the channel buffer in one clock cycle. For example, in Figure 4.7, the *read-pointer* is currently pointing location 7 and it should point to location 11 as this is the next location that is occupied. When some locations of the channel buffer are occupied, the *read-pointer* should only points to those locations one by one per clock cycle. We propose a 4-bit fast buffer read circuit to generate a *read-pointer* (RD) as illustrated in Figure 4.10.

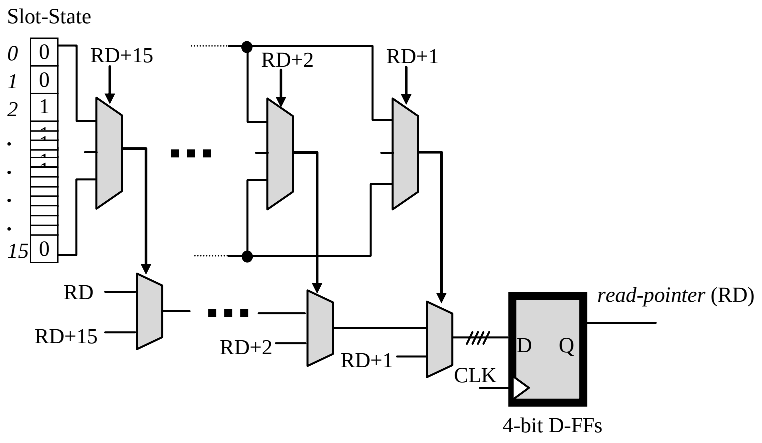


Figure 4.10: 4-bit EDVC Fast *Read-Pointer*

The RD only counts and produces the address of occupied locations of the channel buffer. For example in Figure 4.7, the *read-pointer* (RD) will count and produce addresses of 11, 1, 4 and 7 for consecutive clock cycles. The output of the *read-pointer* circuit (*RD*) points to the initial location of *Slot-State* table that is set and remains there for one clock cycle. During the same clock cycle, the state of current flit location is determined whether it is blocked or not. Regardless of the flit condition, the *read-pointer* advances to the next occupied location at the next clock. Similarly, it will continue to access flit-occupied locations per clock cycle. Considering the buffer locations 1, 4, 7 and 11 are occupied as shown in Figure 4.7, starting from location 7 the *read-pointer* accesses the other three locations in the following three clock cycle one by one. It does not matter whether one or more of the flits at location 1, 4, 7 and 11 are blocked. The analysis shows that the *read-pointer* does select occupied slots with the same priority. In fact, there is no priority and any flit in the buffer will be pointed by the *read-pointer* for one clock cycle one by one. It doesn't matter whether the occupied slot flit is blocked or not, the *read-pointer* stays at each occupied slot for one clock cycle.

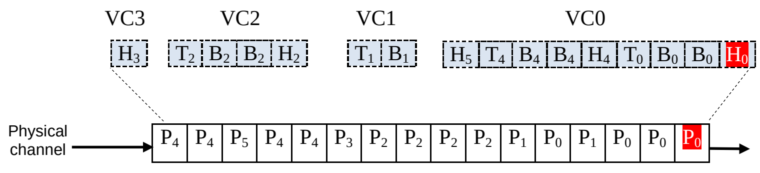
4.4 Novelty of EDVC Mechanism

The virtual channel mechanism for wormhole routing is a common approach used in many NoC designs [24, 25, 50]. When the header flit of a packet enters a VC buffer, the packet reserves that specific VC. The reservation of a VC is maintained until the tail flit enters. At this point, the VC can accept a new packet if it has free space. In this way, a VC can contain two parts of two packets at a time. Assume that a VC has two parts of two packets simultaneously. The first packet can block the second packet in spite of the fact that the route of the second packet is open. It is commonly known as HoL (Head of Line) blocking. HoL blocking leads to a number of problems in NoC systems such as congestion, deadlock, and monopoly of one VC over the whole input-port buffer space.

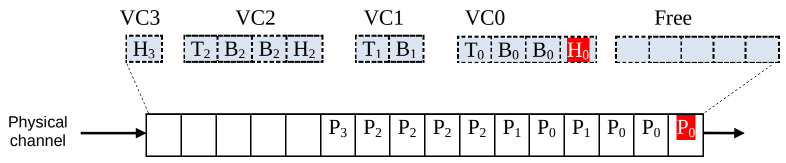
As mentioned in Section 4.2.2, a common feature of VC-based mechanisms such as LLD is the closing of a VC when a packet head flit associated with the VC faces a blockage. When the head flit of a VC faces with a blockage, the arbiter sends a *VC-block* signal to the input-port to prevent the VC to generate request signal (see Figure 3.13). In EDVC mechanism, the *VC-block* signal also closes the blocked VC (by activating its *VC-full* signal). This blocking process travels back through the routers that are related to the blocked VC and leads the associated VCs to be closed too. In short and in terms of functionality when compared with LLD, the LLD mechanism blocks a VC when it is full, and in the case of our EDVC a VC is blocked when it is full or if it is blocked in the downstream router indicated by the corresponding *VC-full* signal from the relevant VC. This behavior of EDVC mechanism leads to some features such as high alleviation of HoL problem, lower buffer space for blocked VCs, and preventing of deadlock without dedicating in default any buffer space for each VC. We further discuss these features in the following sections.

4.4.1 VCs for Blocked and Unblocked Packets

The probability of obtaining a free VC for the unblocked packets in EDVC is much higher when compared with LLD. For better understanding, we compared two situations. Figure 4.11a represents a situation for LLD, where packets P4 and P5 remain blocked until the packet P0 becomes unblocked. Figure 4.11b represents the EDVC situation where the blocked packet P0 causes the VC0 stops receiving flits, so the packets P4 and P5 can occupy the buffer when one of the VC1, VC2 or VC3 becomes free.


VC3
VC2
VC1
VC0
H_3
T_2 B_2 B_2 H_2
T_1 B_1
H_5 T_4 B_4 B_4 H_4 T_0 B_0 B_0 H_0
Physical channel
P_4 P_4 P_5 P_4 P_4 P_3 P_2 P_2 P_2 P_2 P_1 P_0 P_1 P_0 P_0 P_0

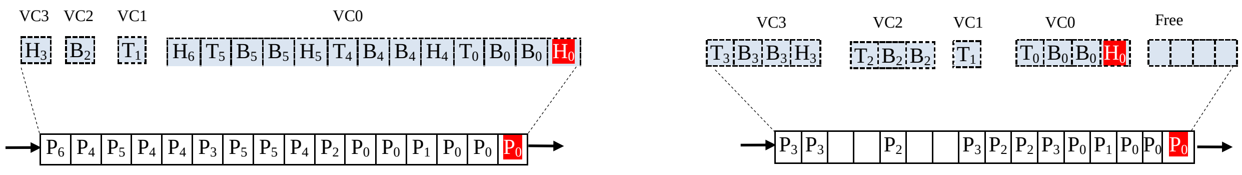
(a) LLD: P0 blockage leads to HOL blocking



(b) EDVC: No HOL blocking

Figure 4.11: Packet Blocking in LLD and EDVC.

In our EDVC approach, when a packet faces a blockage, its VC will utilize a minimum space of the input-port buffer. Consider a situation of a blocked packet in the VC. In the case of traditional LLD, the upstream router continues to send new packets to the VC that will allocate more buffer space to accommodate these packets leading to the allocation of a large but useless buffer space as illustrated in Figure 4.12a. However, the new packets in our EDVC approach remain in the upstream router until a downstream VC becomes empty. In fact, more free space for other unblocked VCs is provided as shown in Figure 4.12b. Consequently, the performance and buffer utilization of EDVC is much higher when compared with the LLD methodology.

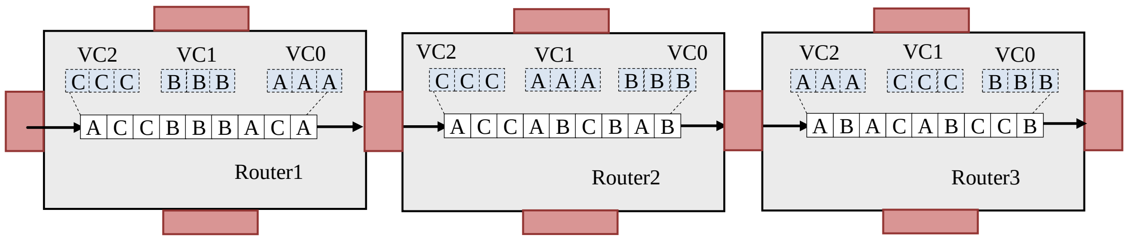

VC3 VC2 VC1 VC0
H_3 B_2 T_1
H_6 T_5 B_5 B_5 H_5 T_4 B_4 B_4 H_4 T_0 B_0 B_0 H_0
P_6 P_4 P_5 P_4 P_4 P_3 P_5 P_5 P_4 P_2 P_0 P_0 P_1 P_0 P_0 P_0
VC3 VC2 VC1 VC0 Free
T_3 B_3 B_3 H_3
T_2 B_2 B_2
T_1
T_0 B_0 B_0 H_0
P_3 P_3 P_2 P_3 P_2 P_2 P_3 P_0 P_1 P_0 P_0 P_0

(a) LLD VC0 has a large buffer space (b) EDVC Buffer is equally divided among all VCs

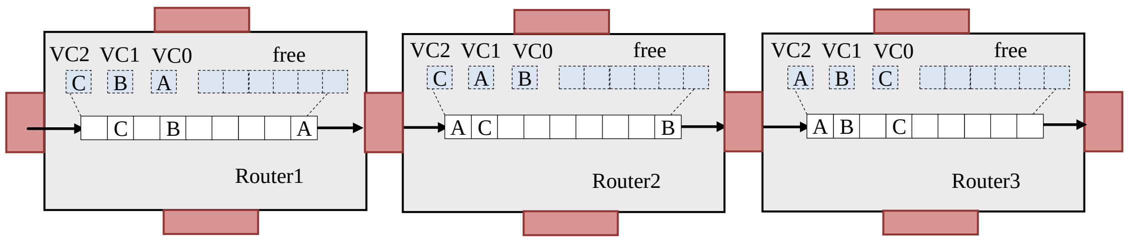
Figure 4.12: (a) P4, P5 and P6 packets are blocked due to HOL of P0 (b) Free slots are used by the other VCs

4.4.2 Lower Congestion

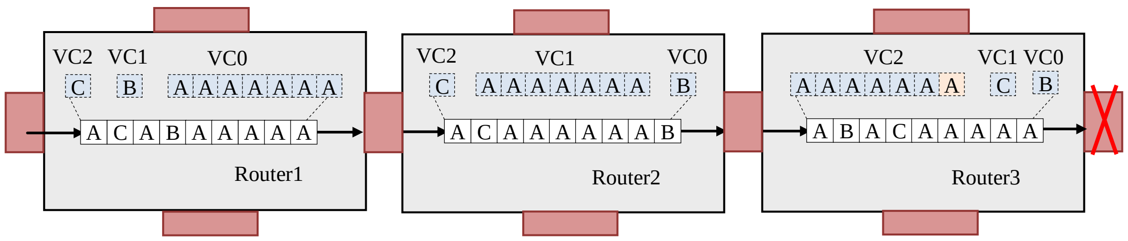
When a packet travels in an NoC, it reserves a VC in each router of its route. The packet flow rate depends on various NoC conditions including scheduling and traffic patterns. For example, in the case of a round-robin scheduling, the packet flow shown in Figure 4.13a is more crowded than in Figure 4.13b due to different traffic conditions. One of the conditions, which leads to congestion in a DAMQ based NoC, takes place when the packets become blocked. Packet blocking at each router causes the packet flow to become congested on its route. For example, Figure 4.13c shows the route of packet *A* as it passes through *Router1* and *Router2* and then becomes blocked in *Router3*. If packet *A* has a large number of flits, it can consume all the buffer space of the input-port in the routers on its path and creates a serious level of congestion.



(a) Packet flow rate is uniform/high for high level of traffic.



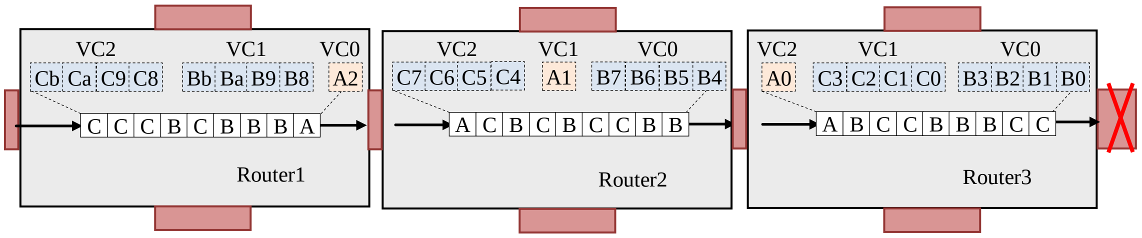
(b) Packet flow rate is uniform/low for low level of traffic.



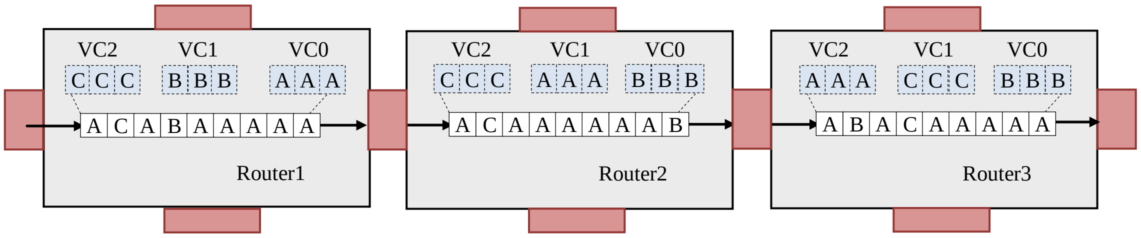
(c) Packet flow is congested due to the blocking of packet *A*.

Figure 4.13: LLD Packet Flow Situations.

As described earlier that in the case of our EDVC mechanism, the *VC-block* signal of upstream router also activates the *VC-full* signal of the related VC. This process spreads back through the routers that are related to the blocked VC as illustrated in Figure 4.14a. In EDVC, each router acknowledges its upstream routers about the state of its VCs. The lowest possible buffer capacity is assigned to the blocked packets. For example, consider the packet flow scenario in Figure 4.14a. Assume that the data flow is in a pipelined communication, the VC requests are issued in a round-robin style, and the acknowledgement is sequential among the routers. As soon as the flit *A0* in *Router3* is blocked, the *Router3* informs the *Router2* at the following clock cycle. Due to round-robin form of VC requests, the *router2* sends the 2nd flit of *A* after sending a flit of packets *B* and *C*. Therefore, the blockage of *A1* is reached sooner than its arbitration. The flit *A1* in *Router2* becomes blocked, and the *Router2* informs the *Router1* at the following clock cycle. The same condition of *A1* occurs for *A2*. In such scenario, the blocking is transferred back in 2 clock cycles per router. In this way, all the upstream routers cease sending the blocked packets during the blockage and the router buffers are assigned to other VCs holding free unblocked packets. As shown in Figure 4.14a, one slot of each router is assigned for packet *A*. When the blockage is removed, the flow of packets returns to normal as depicted in Figure 4.14b resulting in high buffer utilization and overall higher NoC performance.



(a) Packet A occupies minimum VC buffer space during blockage.



(b) Uniform packet flow after the blockage of packet A is removed.

Figure 4.14: EDVC Packet Flow Situations.

Consider the above conditions for ViChaR. The ViChaR cannot theoretically reserve a specific room for each VC, and its VC size varies from one to the maximum size of the channel buffer [13]. Therefore, in some cases this mechanism will create a deadlock or a high traffic contention. ViChaR dynamically allocates VCs and grants new flit on a first come first served basis and there is no priority for the new packets. In the case of blocking, a packet can occupy all the slots of a channel buffer and thus prevents any new packet to pass through the router. If this blocking continues, the packet will occupy all of the upstream routers and no other new packet will be able to pass through the route. This blocking can spread in the entire NoC leading to a deadlock.

4.4.3 Deadlock Avoidance

Another feature of EDVC is deadlock avoidance. We illustrate this feature by way of a deadlock scenario in traditional LLD mechanism. Assume that there is no reserved space dedicated for each VC in the LLD mechanism. The packets destined for a specific VC may occupy the entire input-port buffer space, and any new packet destined to this port may not be able to gain access of the input-port buffer [15]. Moreover, sharing channels and buffers increase the probability of packet blocking exponentially that will lead to more contention and deadlock like conditions for the NoC communication. Figure 4.15 shows a deadlock like situation for the LLD mechanism in a 4×4 NoC. Assume there are four flits per packet, two VCs per input-port, two slots per buffer, XY routing, round-robin scheduling, and packets are sent by the source cores S_1, S_2, S_4 and S_5 to the destination core D_{10}.

When the first flits of all the source cores are injected in the NoC, packets P_2 and P_5 can reserve a complete path to their destination D_{10} due to the availability of two VCs in the northern input-port of router#10 and their flits can reach D_{10}. However, packets P_1 and P_4 can only reserve a path up to router#6. After the injection of second flits of all the sources, P_1 and P_4 will occupy all the buffer space of north and west input-ports of router#6. When the third flits of all the

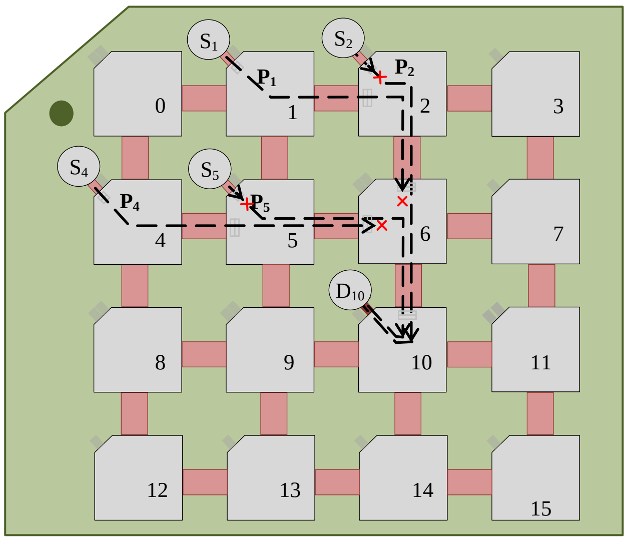


Figure 4.15: High Contention Situation in a LLD NoC

source cores are injected, a deadlock will occur. The first flits of packets P_1 and P_4 cannot advance in the NoC because the two VCs of north input-port of router#10 is already reserved by P_2 and P_5, and the third flits of packets P_2 and P_5 cannot advanced in the NoC because the north and west input-port buffers of router#6 are already occupied by packets *P_1* and *P_4*. However, this problem would not occur in our EDVC mechanism. In fact, when the first flits of *P_1* and *P_4* are blocked in router #6, their associated input-ports cease receiving those packets. In this way, there will be free space for *P_2* and *P_5* to pass the router#6 and reach the destination *D_{10}*. This section also illustrates another feature of EDVC approach that it prevents deadlock without dedicating in default any buffer space for each VC.

4.4.4 Novel EDVC based VC Organization

The novelty of our EDVC approach can be summarised as follows.

- We have introduced a new DAMQ-based input-port architecture that improves NoC performance considerably by adding a little hardware.
- The EDVC mechanism employs logic circuits instead of tables to manage shared VC slots and to determine the next free write-slot and available read-slot. It saves one clock cycle for each flit arrival/departure from input-port.
- Our EDVC approach is much simpler as compared to table-based DAMQ mechanisms such as LLD [14, 18] or ViChaR [13]. The main component of EDVC is the pointer circuits of Figures 4.9 and 4.10, which has a scalable structure to optimize its design and timing performance.
- There is no configuration constraints in our EDVC approach as compared to other DAMQ mechanisms, and it can work in any configuration. For example, the LLD based approaches require a reserved space for each VC, and the ViChaR buffer grows bigger with the increasing number of flits per packet.
- The EDVC approach employs a simple congestion avoidance mechanism.

4.5 Experimental Results

In this section, we compare the hardware requirement and performance of our EDVC mechanism with traditional Link-list based LLD and ViChaR mechanisms. Three types of EDVC organizations, Fast-Read, Fast-Write, and Fast-Read/Write are evaluated. The Fast-Read architecture employs a simple write-pointer and fast *read-pointer* shown in Figures 4.6b and 4.10 respectively. The *read-pointer* of Fast-Write architecture is a simple counter (see Figure 4.6a); while its write-pointer employs a fast write mechanism, as shown in Figure 4.9. The Fast-Read/Write architecture utilizes the fast write-pointer and fast *read-pointer* illustrated in Figures 4.9 and 4.10 respectively.

4.5.1 Hardware Requirements and Parameters of Input-Ports

The hardware requirements and characteristics of EDVC Fast-Write, Fast-Read/Write, LLD and ViChaR input-port architectures are determined by employing Synopsys Design Compiler for generic 90nm technology and Mentor Graphics ModelSim for Stratix-III FPGA. We have coded the micro-architectures using Verilog and simulation is performed by employing the ModelSim to measure various hardware metrics. The Synopsys design compiler is used to measure the power consumption and area of NoCs. We measure the hardware metrics of the input-ports of EDVC, LLD and ViChaR routers. All the ports use a dual-ported SRAM for data buffering. We apply the same setup constrains to all the input-ports. The setup for the input-port ASIC power and area employs CMOS technology parameters of Synopsys Generic 90nm Library, global operating voltage of 1.2V and time period of 5nsec (200MHz). The width of slot buffer is equal to the flit size of 16 bits.

The characteristics of input-port micro-architecture are listed based on their buffer sizes in Table 4.2. The details of the other modules including arbiter and crossbar are also listed at the end of table. The EDVC Fast-Write input buffer has the optimum area and timing characteristics among all the input-ports. On average, the EDVC Fast-Write consumes 15% less IC area, 58% less power consumption, 64% less critical path delay, 4% less combinational logic elements and 15% less registers as compared to LLD. For FPGA implementation results, the table also provided the amount of SRAM based registers in the brackets. EDVC Fast-Write uses a simple *read-pointer* (Figure 4.6a) versus a fast *read-pointer* (Figure 4.10) employed in EDVC Fast-Read/Write approach. The simple *read-pointer* increments per clock cycle regardless of the flit existence in the input-port buffer that leads to 9% higher power consumption of Fast-Write versus Fast-Write. In spite of optimum hardware characteristics, the EDVC Fast-Write and Fast-Read has lower performance than EDVC Fast-Read/Write that is also investigated in this section. The LLD and ViChaR architectures are table based designs where a register-based control table directs the mechanisms. Registers and/or latches are updated on one edge (or level) of clock causing the table-based mechanisms like LLD and ViChaR to take one clock cycle longer than our EDVC approach that uses dedicated logic circuits.

Table 4.2. Hardware Specification of EDVC, LLD & ViChaR Input-Ports

Type of input-port	ASIC design (90 nm Generic Library)			FPGA design (Altera Stratix III)		
	Total Cell Area *(μm^2)*	Power *(μW)*	Critical path delay *(ns)*	Comb. Logic elements	Registers *(bits)*	f_{max} *(MHz)*
LLD 4-slots	5809	218	1.57	95	112(64[b])	352
ViChaR 4-slots	6646	319	1.56	75	132(64[b])	300
EDVC Fast-Read/Write 4-slots	4991	106 *(60[a])*	1.11	83	107(64[b])	384
EDVC Fast-Write 4-slots	4674	108	0.57	78	97(64[b])	1082
LLD 8-slots	10328	370	1.54	180	204(128[b])	286
ViChaR 8-slots	21274	1236	2.26	306	392 (128[b])	197
EDVC Fast-Read/Write 8-slots	9524	150 *(88[a])*	1.62	206	186(128[b])	244
EDVC Fast-Write 8-slots	8687	162	0.47	174	174(128[b])	851
LLD 16-slots	19813	688	2.02	332	388(256[b])	271
ViChaR 16-slots	48463	2968	2.76	548	896 (256[b])	175
EDVC Fast-Read/Write 16-slots	19448	240 *(147[a])*	2.22	441	340(256[b])	171
EDVC Fast-Write 16-slots	17016	263	0.92	324	327(256[b])	726
LLD 32-slots	39174	1306	3.23	670	764(512[b])	218
ViChaR 32-slot	109849	6989	2.86	1183	2040 (512[b])	125
EDVC Fast-Read/Write 32-slots	40716	413 *(262[a])*	4.64	964	646(512[b])	118
EDVC Fast-Write 32-slots	34295	457	0.94	727	632(512[b])	597
Arbiter 4-VC	28380	1904	4.17	1116	240	127
Cross-bar	2502	611	-	160	-	-

[a] Power consumption at 100 MHz, [b] SRAM registers

The ViChaR architecture is expensive in terms of hardware cost among all the other architectures. The higher cost of ViChaR is due to large control table (even bigger than that of LLD). The following equations show the Size of LLD and ViChaR Control Tables (SCT).

$$SCT_{LLD} = SB.\mathit{ln}(SB)+ SB+VC + 2.VC.\mathit{ln}(SB) \qquad (1)$$

$$SCT_{ViChaR} = SB.\mathit{ln}(SB).FP + SB + 2.SB.\mathit{ln}(FP) \qquad (2)$$

Where

- SB is the number of slots per port buffer.
- FP is the number of flits per packet.
- VC is the number of virtual channels.

Assuming a configuration for the experiments where VC=4, SB=16, and FP=16, the SCT_{ViChaR} is almost 10 times of the SCT_{LLD} (VC = SB in ViChaR). The only advantage of ViChaR is its lower critical path delay for 32-slot buffer. This is due to the lower dependency among the cells of its control table. In other words, the address location of each stored flit in the input-port buffer is recorded in the control table, and there is no link among these addresses. However, a large control table of ViChaR can lead to a lower f_{max} (maximum frequency) than all the other input-ports.

For larger 32, 16 and 8-slot input-buffers, EDVC Fast-Read/Write has lower f_{max} for FPGA implementation and higher critical path for ASIC design as compared to LLD. It is due to the fast *read-pointer* module that grows bigger with the size of the input-port buffer. The critical path of EDVC Fast-Read/Write includes the fast *read-pointer* having a large number of Multiplexers. When the size of input-port buffer increases, the multiplexing stages of fast *read-pointer* grows that will in-turn increase the critical path delay for EDVC Fast-Read/Write and Fast-Read. There are different ways to optimize the critical path of EDVC Fast-Read/Write. First of all, the critical path delay of an input-port can be ignored when the arbiter critical path delay is larger than that of the input-port. For example, the critical path of a 4-VC arbiter in our implementation is longer than 4, 8 and 16-slot EDVC Fast-Read/Write input-ports (as listed in Table 4.2). An optimal design of fast *read-pointer* will also lead to lower critical path delay for EDVC Fast-Read/Write input-port. We will present optimal architectures for fast *read-pointer* and write-pointer in the following chapter. Another timing advantage of EDVC approach should also be kept in mind that it takes one less clock cycle for flit arrival/departure than LLD approach.

The EDVC Fast-Read/Write based input-port shows a significant saving in the power consumption and register usage as compared to ViChaR and LLD input-ports. As one can observe from Table 4.2, EDVC Fast-Read/Write, on average consumes 61% less power and 10% less buffer as compared to LLD. The switching associated with the control table (Registers) is larger causing a higher power consumption of LLD and ViChaR as compared to our EDVC Fast-Read/Write. We expect that as the input-port buffer depth is increased, the Fast-Read/Write input-port hardware area becomes lower than those of LLD input-port. However, the Fast-Read/Write hardware area increases more than the LLD area of input-port for 32-slot buffer. This increase is due to the impact of the Fast-Read/Write-pointers logic illustrated in Figures 4.9 and 4.10.

4.5.2 EDVC Performance Evaluation

1) Application Specific and Hotspot Traffic Patterns

In the first experiment, we simulated and tested EDVC mechanism for two different traffic patterns including Application-Specific and Hotspot traffic patterns [53, 54]. For Application-Specific traffic, two NoC applications presented in Section 3.6.2: MPEG4 Decoder and Audio-Video (AV) benchmarks are tested for 3×4 and 4×4 mesh topology NoCs as given in Figures 3.20 and 3.21 respectively. For Hotspot traffic, one destination is chosen for all the source cores during a time period. By evaluating the results of these three traffic patterns, we demonstrate the efficiency of our EDVC approach in terms of throughput and latency of the NoCs.

The NoC topology selected is either 3×4 or 4×4 mesh, and packet communication follows the XY routing as shown in Figures 3.20b and 3.21b. The communication of packets is based on wormhole switching where the channel width is equal to the flit size (16 bits). Each packet is made of sixteen flits. Each input-port utilizes 2 VCs and 8 slots per input-port buffer in the Application-Specific traffic. In the case of Hotspot traffic, each input-port has 4 VCs and the buffer depth is varied from 4 to 32 slots. The link delay between two routers is negligible as compared to the delay of a router and it is ignored. All the source/destination cores and routers operate at the same clock rate. In the case of Hotspot traffic pattern, all the source cores send their packets to one destination (e.g. destination core#10 of Figure 4.15).

We measured throughput and latency where throughput is measured by the rate of packets received to the maximum number of packets being injected at a specific time. The average latency is measured by the average time delays (per clock cycle) associated with the departure and arrival of a specific number of packets in the NoC. We apply different packet injection rate to measure the performance in the Application-Specific traffic. Packet injection rate is changed per time unit. The time unit is determined based on the maximum bandwidth of the source cores. For example, Source Core#8 in the MPEG4-decoder has a maximum bandwidth of 1580 flits. Therefore, the time unit will be 1580 clock cycles (assuming one flit per clock injection by each source core). The same measurement is performed for the AV application i.e. Core#14 has a maximum bandwidth of 192078 flits for a time unit of 192078 clock cycles.

The results of Figures 4.16- 4.19 provide throughput and latency for MPEG4-decoder and AV-benchmark applications. The throughput and latency of Fast-Read/Write is almost 100% and

50% better than those of LLD mechanism. This improvement is due to the fact that there is a little contention in the data flow, where the only determining factor is the speed of routers. As mentioned earlier, the passage of a flit through a LLD router takes 4 clock-events where it takes two clock-events for an EDVC router as illustrated in Figure 4.3. Flit arrival/departure in EDVC router is one cycle (two clock events/edges) as compared to two cycles for LLD routers. Moreover, the contention is low because the destination of each source is fixed, and most of the source cores send a few flits per time unit.

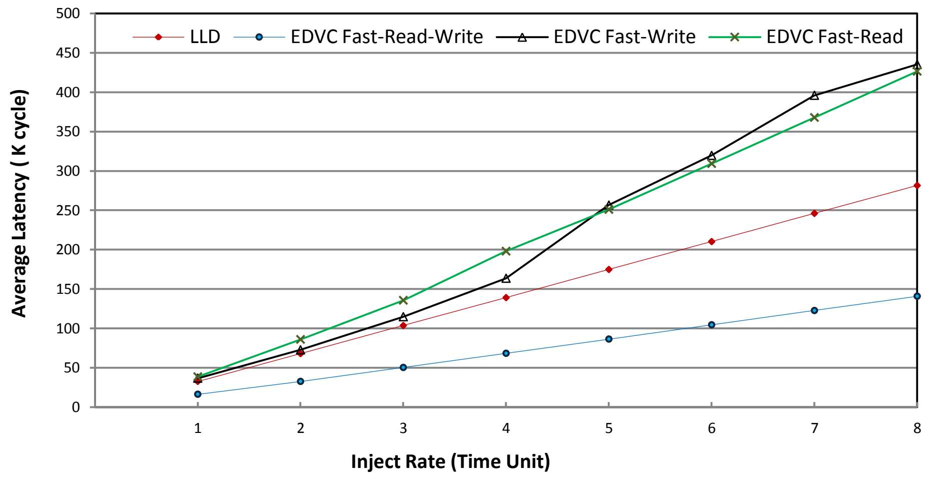


Figure 4.16: Average Latency for MPEG4 Decoder Traffic

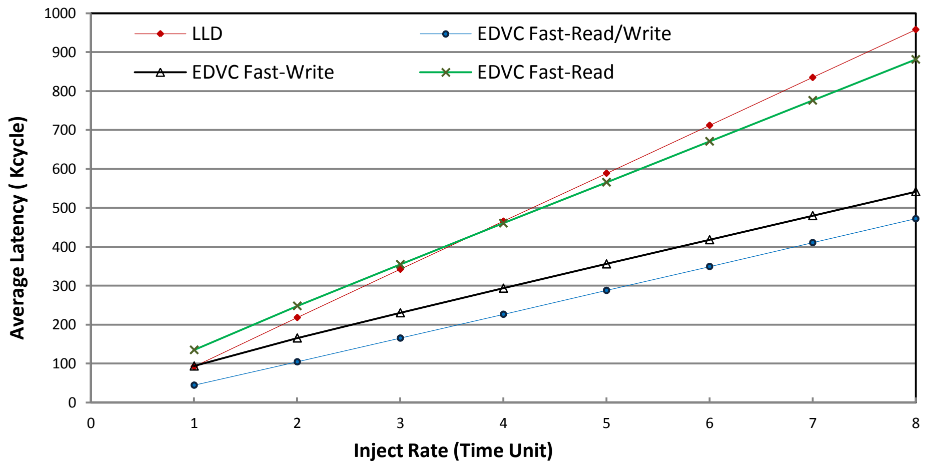


Figure 4.17: Average Latency for AV Benchmark Traffic

EDVC Fast-Read/Write has lower latency than that of LLD for all the injection rates as shown in Figures 4.16 and 4.18. However, it is higher for the EDVC Fast-Read or Fast-Write. For MPEG4, seven source cores send data to one destination core#5 causing high contention. The higher contention results in higher latency for EDVC Fast-Read and Fast-Write as compared to LLD. For AV benchmark, the NoC communication is less congested (e.g. a maximum of 4 cores send packet to destination core #10). Due to which, the latency is lower in EDVC Fast-Write for all the injection rates and for higher injection rates as compared to the latencies for EDVC Fast-Read. A slower write to an input-port buffer of EDVC Fast-Read router affects all the downstream input-ports that may intend to send packets to that router. However, a slower read from an input-port buffer of an EDVC Fast-Write router will only affect one input-port that is itself.

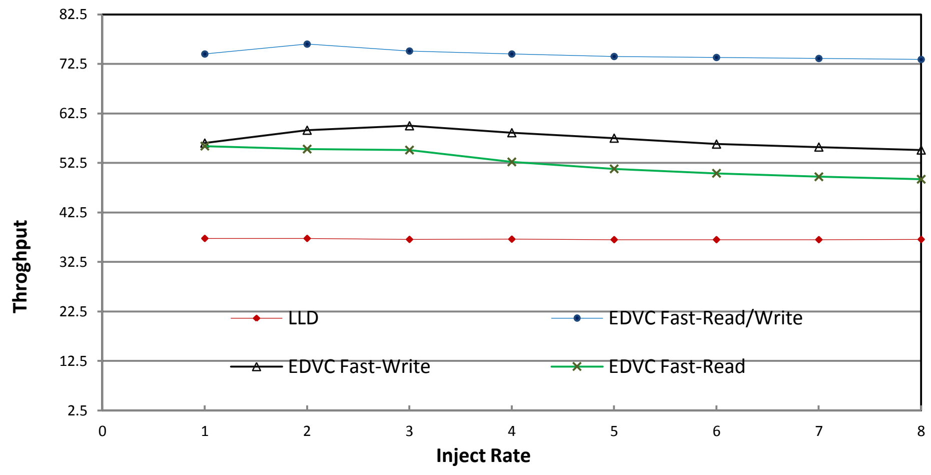


Figure 4.18: Average Throughput for MPEG4 Decoder Traffic

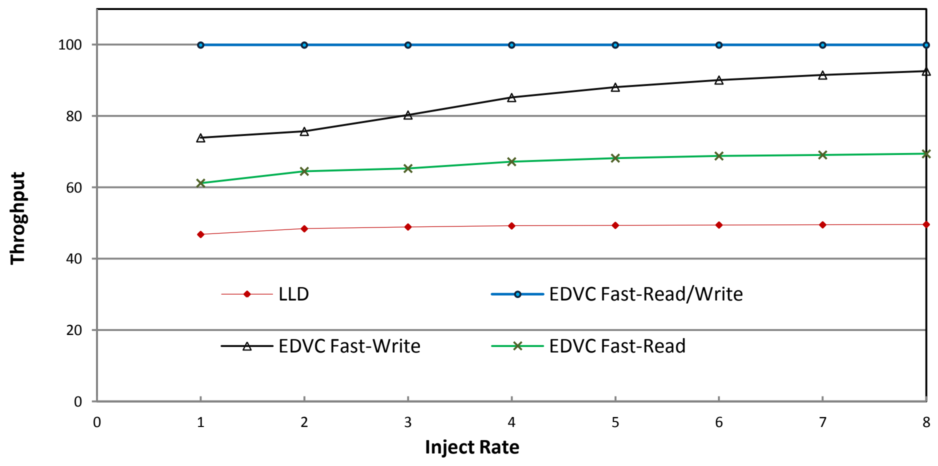


Figure 4.19: Average Throughput for AV Benchmark Traffic

In the case of Hotspot traffic, we compared our EDVC approach with LLD as well as ViChaR approaches. The performance of EDVC mechanism is better than LLD and ViChaR approaches as shown in Figures 4.20 and 4.21. In fact, all the EDVC mechanisms including Fast-Read, Write and Read/Write has higher throughput and lower latency than those of LLD and ViChaR. This is due to the fact that one destination is chosen for all the source cores and traffic becomes congested creating large number of packet blockages. The average throughput and latency of the Fast-Read/Write approach is 100% higher and 48% lower (respectively) than those of LLD and ViChaR. The high contention causes the performance of EDVC Fast-Read and EDVC Fast-Write becomes lower than EDVC Fast-Read/Write. The simple write-pointer and *read-pointer* are very slow when the input-port buffer is crowded (Figure 4.6).

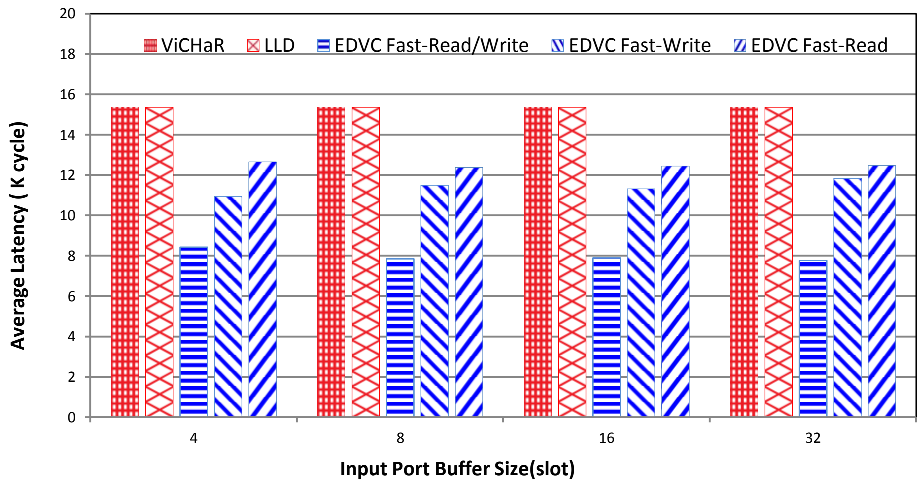


Figure 4.20: Average Latency for Hotspot Traffic

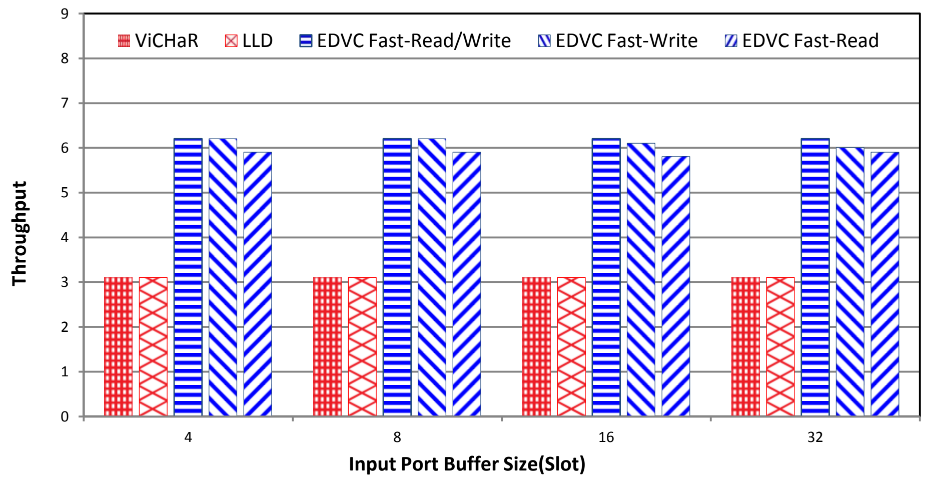


Figure 4.21: Average Throughput for Hotspot Traffic

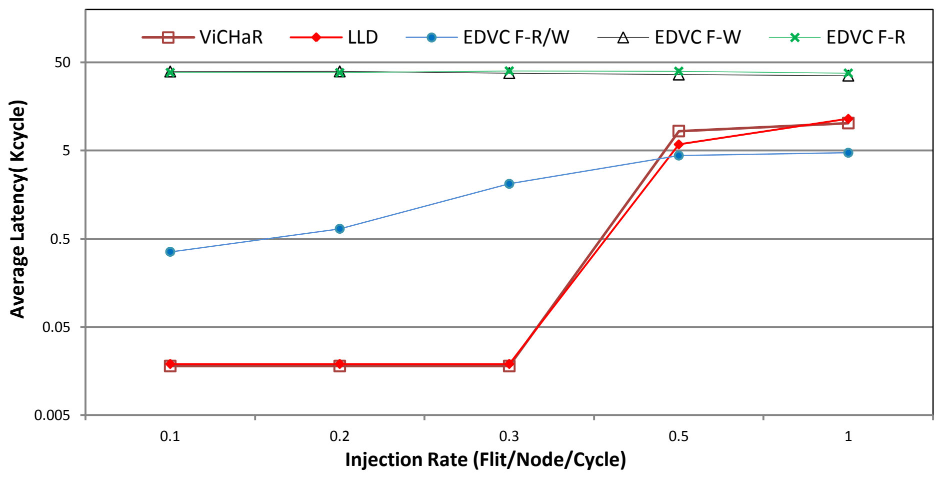


Figure 4.22: Average Latency for Tornado Traffic.

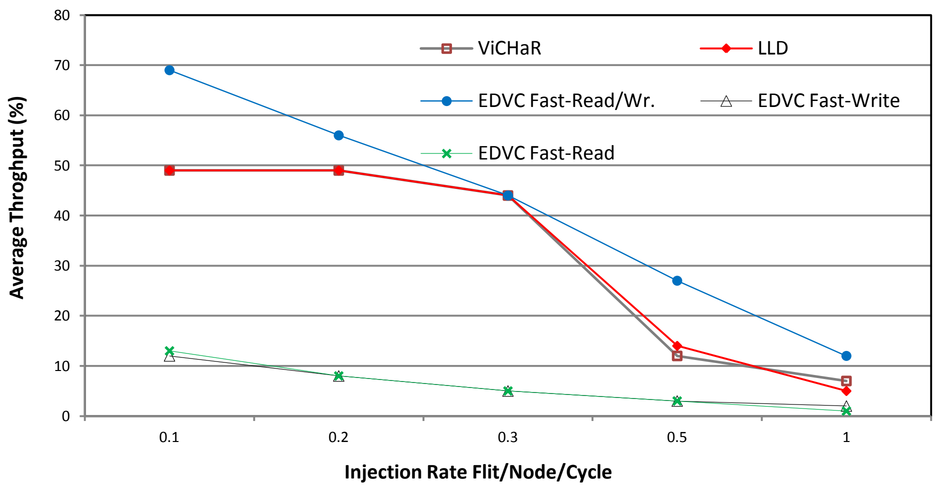


Figure 4.23: Average Throughput for Tornado Traffic.

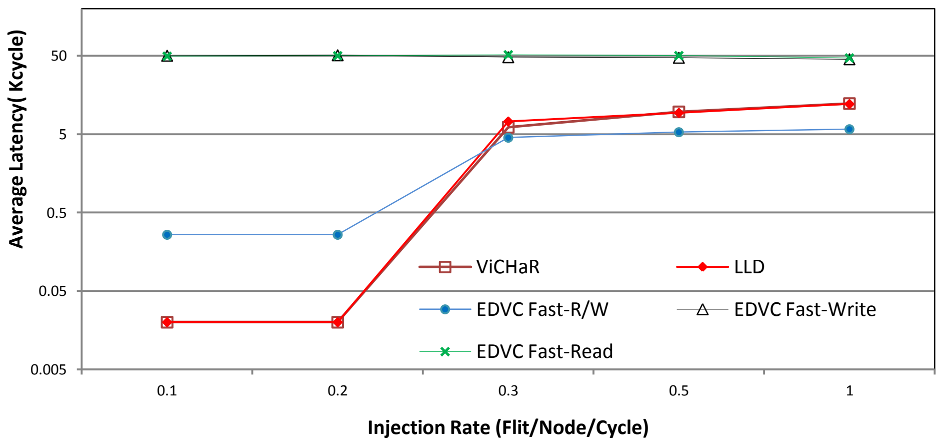


Figure 4.24: Average Latency for Complement Traffic.

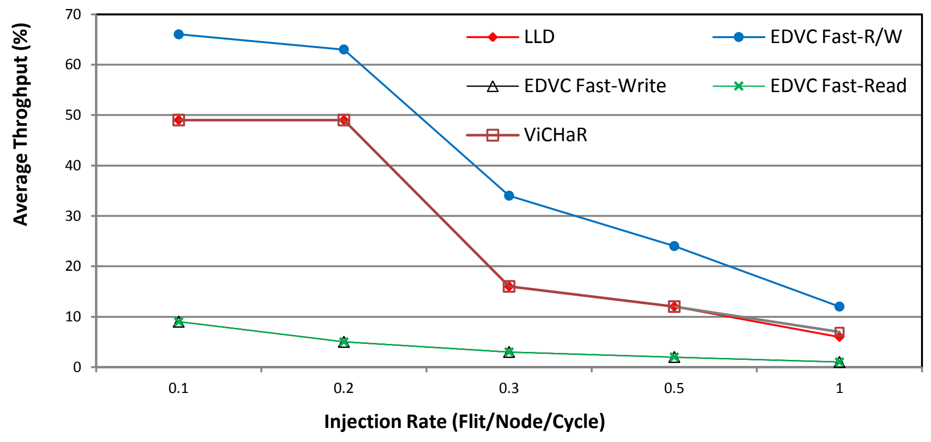


Figure 4.25: Average Throughput for Complement Traffic.

The performance results also indicate a faster EDVC Fast-Write as compared to Fast-Read. This is due to a slower write to the input-port buffer for an EDVC Fast-Read router that affects to all the downstream input-ports. However, a slower read from an input-port of EDVC Fast-Write router will only affect one input-port that is itself. The throughput and latency improvements for Fast-Write are 4% and 10% better than the Fast-Read mechanism. In spite of higher VC numbers of ViChaR (VC number is equal the slot numbers), the ViChaR and LLD have the same performance as both ViChaR and LLD use control tables for their dynamic VC mechanism i.e. both have the same flit arrival/departure delays. Moreover, in the mesh topology, there are a maximum of 4 requests for an output-port at each clock cycle and four VCs are adequate for the NoC communication.

2) Performance Analysis for 8×8 NoC Topology

In the 2nd experiment, we explore and compare our EDVC approach for a larger 8 × 8 mesh topology and for some other commonly used traffic patterns such as Tornado and Complement [16, 55]. Tornado and Complement traffic benchmarks create high contention traffic uniformly in an NoC. For an $m \times m$ mesh topology, source address (Sx, Sy) where $0 \leqslant x, y \leqslant m-1$, and a destination address (Dx, Dy) is determined by the following equations for Tornado and Complement traffic:

For Tornado: $Dx = Sx+(m/2)-1$, $Dy = Sy+(m/2)-1$ (3)

For Complement: $Dx = m-Sx-1$, $Dy = m-Sy-1$ (4)

In XY routing and Tornado traffic, all the routers are uniformly crowded, where in the case of Complement traffic the side routers are more crowded than the middle. A packet consists of 16 flits, and each input-port includes one central 8-slot buffer. There are 4 VCs per each input-port except for ViChaR that has 8 VCs to prevent deadlock in ViChaR communication. The LLD mechanism reserves at least a slot per VC to prevent deadlock in its communication. The performance metrics are measured per flit injection rates as illustrated in Figures 4.22- 4.25. The results show higher performance for EDVC Fast-Read/Write in high injection rates. For example, the average latencies of Fast-Read/Write at 0.3, 0.5 and 1 injection rates are 46% less than those of LLD for Complement traffic. The average throughput of EDVC Fast-Read/Write is higher than those of LLD in both patterns i.e. 29% and 50% higher in Tornado, and Complement traffics respectively.

For higher injection rates, the NoCs become populated with higher rate of contention. As previously discussed, EDVC improves NoC performance in high contention. EDVC mechanism reduces the probability of monopolizing an input-port by a growing VC. Another feature of EDVC is the prevention of congestion by propagation of VC-full signal from the remote routers along the routing path. The LLD and ViChaR show similar performance and their similarities are also due to the same flit arrival/departure delays.

We expect two times better throughput and lower latency for EDVC Fast-Read/Write at low injection rates (assuming no contention) as compared to LLD or ViChaR due to two times faster clock for EDVC Fast-Read/Write NoC. However, the movement of flits is a bit slower in Fast-Read/Write mechanism than that of LLD or ViChaR mechanisms. The slow flit movement is due to the hardware/technique used for handling blocked VC that is discussed and presented in Section 4.2.3. A blocked VC becomes free to be read when the read pointer points to the first location of the input-port buffer, and it becomes free to be written when the read pointer points to the last location of input-port buffer. The slower movement of flit for EDVC Fast-Read/Write leads to a bit higher latency, however, the throughput is higher as compared to LLD or ViChaR for lower injection rates as illustrated in the results given in Figures 4.23 and 4.25. A bit higher latency of EDVC Fast-Read/Write is shown in Figures 4.22 and 4.24 for injection rates of 0.1 and 0.2. However, for the same injection rates of 0.1 and 0.2, EDVC Fast-Read/Write throughput is 32% and 28% higher when compared with LLD/ViChaR for Tornado and Complement traffic patterns respectively. In the following chapter, we improve the EDVC architecture to illustrate higher performance in low injection rate too.

The EDVC Fast-Read and Fast-Write have the lowest performance and their weakness is due to slower read and write scenario due to simpler/cheaper counter implementations for read/write pointers, discussed earlier in Section 4.3. The slow read and write happens repeatedly for Tornado and Complement Traffic patterns, and eventually leads to the lowest performance of EDVC Fast-Read and Fast-Write. However, the number of slow read and write do not happen often for AV and MPEG4 benchmark experiments that leads to better performance of Fast-Read and Fast-Write mechanisms. In the case of Hotspot traffic, the contention is high and only one destination receives all the data. Therefore, all the flits have to wait around that destination core to be served. A slower flit movement speed in the NoC under such condition does not improve the performance. In other words, the very slow receiving of flits by a destination causes the slow movement of flits that leads to a low performance of EDVC Fast-Read and Fast-Write mechanisms. However, the two times faster clock cycle of Fast-Read and Fast-Write mechanism causes a bit higher performance than LLD mechanism as indicted by the results shown in Figures 4.20 and 4.21.

4.6 Concluding Remarks

The micro-architecture of conventional DAMQ input-port for NoC routers consists of complex and large number of control modules that lead to higher pipeline stages and latency in the NoC. To remedy the drawback of DAMQ based NoC routers we have introduced a few DAMQ input-port architectures having simple yet novel mechanisms requiring lower hardware resources. The micro-architecture of EDVC input-ports are presented and compared with two conventional (table-based) DAMQ input-port designs. The advantage of our EDVC mechanisms are investigated and highlighted. The evaluation results presented support the efficiency of our EDVC mechanism in terms of both performance and hardware overhead. An EDVC input-port consumes on average, 10% less registers for FPGA design and 61% less power for ASIC design. It also has 10% higher f_{max} (maximum frequency) as compared to the LLD input-port implementation for small buffer sizes (e.g. 4 slots). Moreover, the EDVC mechanism improves the latency and throughput by 48-50% and 100% respectively as compared to LLD approaches in the case of Application-Specific traffic in the NoC system. EDVC mechanism also shows better performance for various traffic patterns when compared with LLD and ViChaR techniques.

Chapter 5

Rapid and Efficient Router Architecture

In this chapter, we present our latest NoC router architecture. The chapter is organized into the following sections. The EDVC input-port architecture presented in the last chapter is further improved in terms of structure and mechanism in Section 5.1. In Section 5.2, we propose a new Round Robin (RR) arbiter architecture. The architecture of switch allocators utilized in NoC routers is discussed in detail in Section 5.3. A novel router architecture is presented in Section 5.4 by accommodating the new input-port and RR arbiter described in Sections 5.1 and 5.2. The experimental results for various NoC metrics related to performance and hardware overhead are discussed in Section 5.5. The main features of the approach are listed in Section 5.6.

5.1 Rapid Dynamic Queue Based Input-Port Structure

The EDVC mechanism described in Chapter 4 has some minor drawbacks. First of all, the fast *read-pointer* and *write-pointer* modules presented in Figures 4.9 and 4.10 grow large with the size of the input-port buffer [55]. For example, if the size of the input-port buffer increases *n* times, the first multiplexing stages of *read-pointer* and *write-pointer* will grow exponentially i.e.

n^2 times that will in-turn increase the hardware overhead and the critical path delay, which has direct effect on the speed of EDVC router. The second drawback of EDVC design is its higher latency at lower flit injection rates. To solve these drawbacks, we propose an improved architecture for EDVC called Rapid Dynamic Queue (RDQ), which is more efficient for various NoC configurations and most of the known NoC traffic situations. Figure 5.1 illustrates the micro-architecture of our RDQ input-port. We compare the EDVC and RDQ input-port micro-architectures given in Figures 4.4b and 5.1 respectively. Three modules of EDVC input-port including the *read-pointer*, the *write-pointer* and the blocking logic illustrated in the upper parts of two architectures are improved. The RDQ input-port buffer is illustrated in two parts as compared to one central buffer in EDVC. We present the details of improved input-port modules of RDQ including rapid read/write pointers and blocking circuits.

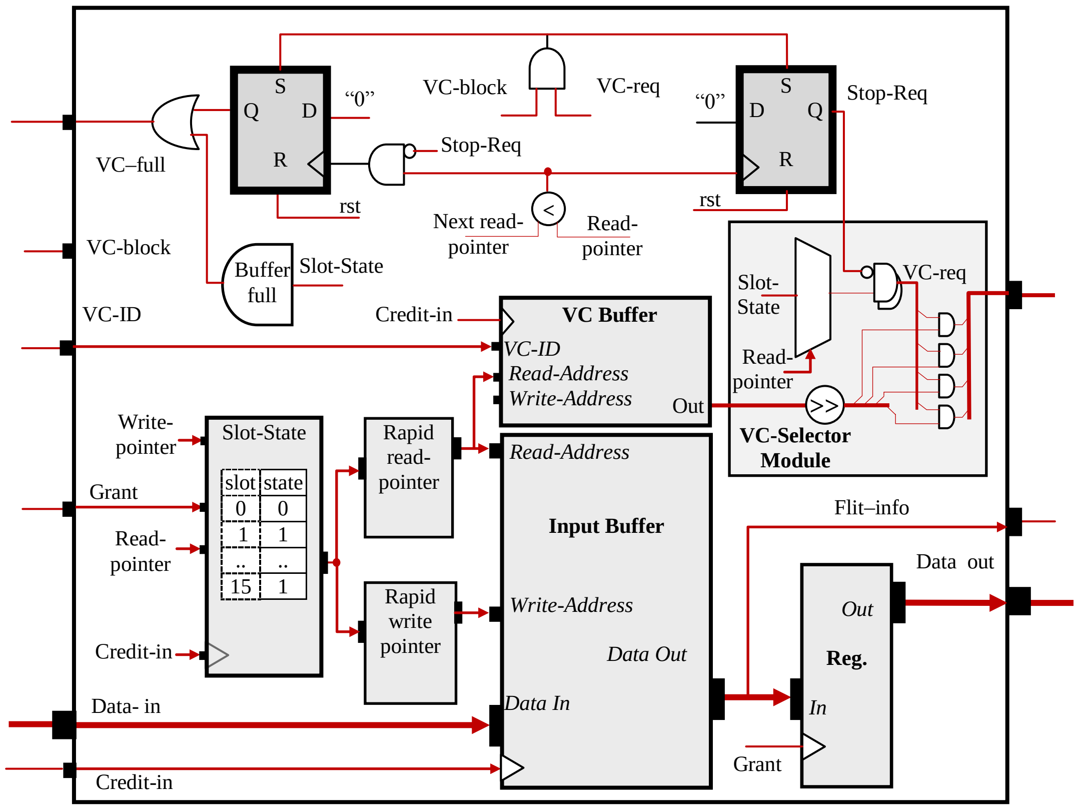


Figure 5.1: RDQ Input-Port Micro-Architecture.

5.1.1 Rapid Read and Write Pointers

As mentioned in Chapter 4, Section 4.3.2 that when some locations of the channel buffer are occupied, the *read-pointer* should only points to those locations. This kind of pointing follows a

Round Robin (RR) priority scheme. In other words, a slot that is just read should have the lowest priority for the next cycle [56]. In this way, each asserted bit of *Slot-State* table is pointed per clock-cycle in an ascending and circular order. Figure 5.2 shows the timing diagram of a 2-bit EDVC fast *read-pointer* (see Figure 4.10) for some scenarios of *Slot-State* content. During the time 1-6 cycles a fixed entry, ‘1111’ is applied that is pointed circularly and bit by bit per clock cycle. At the 6th cycle, the content of *Slot-State* table is changed to ‘1101’ that means the second slot of input buffer becomes empty. At 7th cycle, the content of *Slot-State* table becomes ‘1001’ and will be unchanged till 12th. During cycles 7th and 12th cycles, the first and last slots of the input buffer are read per clock cycle.

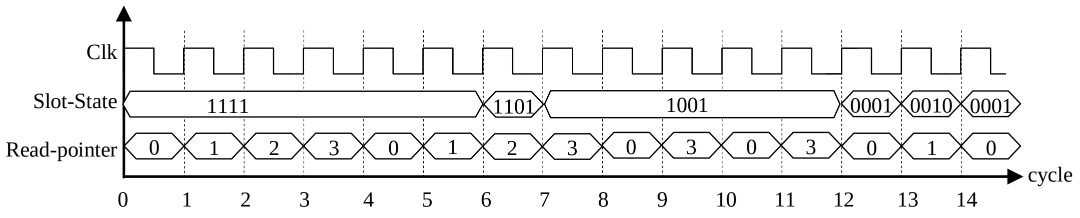


Figure 5.2: Timing diagram of a 2-bit EDVC fast *read-pointer* for some *Slot-State* entries.

We present the micro-architecture of a novel rapid *read-pointer* that follows RR scheme, and it is illustrated in Figure 5.3. Assume S*x* represents the state of an *x*th bit of the *Slot-State* table where x∊ {0, 1…15}, and M*x* represents the *x*th multiplexer of 16 multiplexers shown in Figure 5.3. The multiplexer, M*x* generates the *x*th address if the S*x* is asserted (the *x*th slot is occupied).

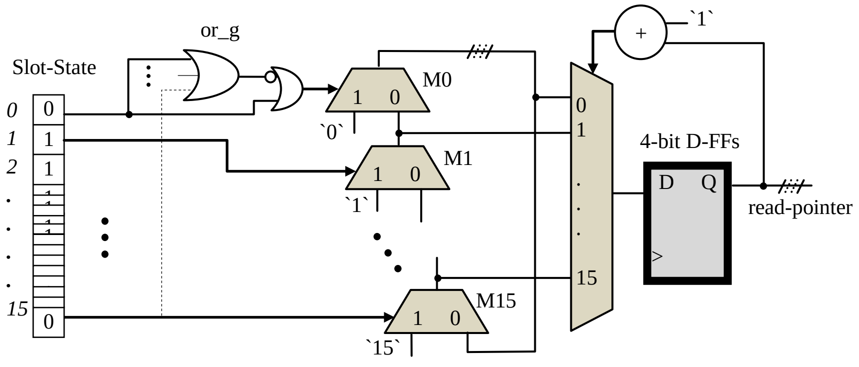


Figure 5.3: Rapid 4-bit *Read-Pointer*

Otherwise, it generates the address of first asserted bit after S*x*. Therefore, by further multiplexing of these multiplexers (i.e. M0 to M15), we can generate the address of first asserted bit after the current pointed address of *read-pointer*. For example, when *read-pointer*=1, the output of multiplexer M2 is selected. M2 generates an address of 2 if S2 is asserted; otherwise, it generates the index of first asserted bit after S2 in a circular order. The OR logic, *or-g* is used to prevent an infinite looping among M0-M15 multiplexers, so that when all the slots of input buffer are empty, the read pointer points to the address zero.

We also propose the rapid *write-pointer* (4-bit) architecture, which is shown in Figure 5.4. If the S*x* is de-asserted (i.e. slot is empty), the *x* address is generated by M*x*. Otherwise, it generates the index of the first de-asserted bit after S*x* in a circular order. Therefore, by further multiplexing with M0-M15 multiplexers, we can choose the address of first de-asserted bit from the current pointed address of the *write-pointer*. For example, when *write-pointer* =1, the output of multiplexer M1 is selected. Then M1 generates a value 1 in cases where S1 is de-asserted, otherwise, it generates the index of first de-asserted bit after S1. The AND logic, *and-g* is used to prevent of infinite looping among the M0-M15 multiplexers when all the slots of input buffer are full. Figure 5.5 illustrates the timing diagram of a rapid 2-bit RDQ *write-pointer* for some *Slot-State* entries.

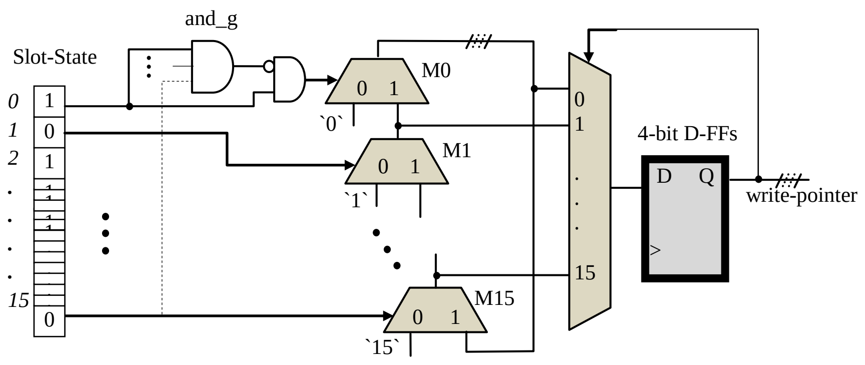


Figure 5.4: Rapid 4-bit *Write-Pointer*

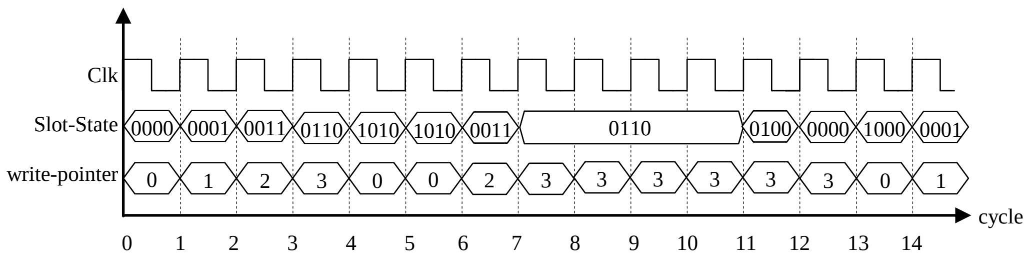


Figure 5.5: Timing Diagram of a 2-bit RDQ *Write-Pointer*.

Our novel rapid *write-pointer* points to an empty slot buffer and will stay there until the slot is written (or occupied). Then it points to the first empty slot in ascending and circular order. The hardware of rapid *read-pointer* and *write-pointer* modules, which are illustrated in Figures 5.3 and 5.4, grow linearly with the size of the input-port buffer. For example, if the size of input-port buffer increases with *n*, the size of first multiplexing stages of Figures 5.3 and 5.4 will also grow with *n*. This characteristics of RDQ pointers leads to a rapid NoC router and we will verify it further in the experimental results section.

5.1.2 RDQ Closing and Requesting Approach (Blocking Circuit)

In the EDVC type mechanism (presented in Chapter 4), the asserted *VC-block* signal causes the VC to stop requesting (*stop-req* signal is asserted) the arbiter and halts to receive of a new flit (*VC-full* signal is asserted). We improve the blocking circuit, which uses *stop-req* and *VC-full* signals by involving one more condition i.e. *VC-req* to assert both the *stop-req* and *VC-full* signals as illustrated in the upper part of Figure 5.1. In this way, we have prevented a condition in data flow when a reserved VC with no data flit becomes closed. In other words, an empty VC cannot make a request, and in case it is reserved by a packet and the packet is blocked (or its *VC-block* is asserted), its *stop-req* and *VC-full* signals cannot be asserted. In summary, an empty VC in any condition cannot be closed for incoming flits. The reason of involving *VC-req* condition is further discussed in this section.

The usage of *VC-req* in the blocking circuit has a direct impact on the performance of RDQ-based NoC router during the low NoC traffic. One AND gate per VC is needed for introducing the *VC-req* condition in the blocking circuit. The blocking signals, *stop-req* and *VC-full* are reset based on the condition of *read-pointer*. At each event, when the *read-pointer* returns to point to the least-significant occupied slot, the *stop-req* signal is reset. This event occurs when the current read address becomes equal or greater than the next read address. It is to be noted that the output and input of *read-pointer* registers represent the current and the next addresses respectively. The *VC-full* signal is de-asserted when the *read-pointer* returns to point to the least-significant occupied slot and the *stop-req* signal is asserted.

Table 5.1 lists the conditions associated with the VC closing and requesting operations. The closing and requesting operations are actuated by the *VC-full* and *VC-req* signals respectively. There are three conditions according to the state of *VC-block*. As mentioned earlier, the *VC-block* becomes set when the VC cannot succeed to win a free VC of the downstream input-port router. We call this VC as a downstream VC in this dissertation. In the first condition, the *VC-block* is reset, so that the closing (*VC-full*) and requesting (*VC-req*) operate in normal condition (see table). The normal condition is when there is no packet blockage in the communication. In the 2nd condition, the *VC-block* becomes set. In case, the *read-pointer* does not point to a flit of the VC, the VC is open to incoming flits. As soon as the *read-pointer* points to a flit of the VC, the *VC-req* switches from 0 to 1 and returns to 0. In other words, the mechanism stops receiving flits for that VC as well as requesting the arbiter. In this condition, the order of VC flits inside the port buffer is kept until the blockage exists. In 3rd condition, the *VC-block* switches from one to zero i.e. the VC can succeed in the arbiter. In case, the *read-pointer* points to a slot that is not the least significant occupied slot of buffer, the VC issues no request to the arbiter and receives no incoming flit. The *read-pointer* continues pointing to the occupied slots until its registers' output becomes greater than its registers' input as illustrated in Figure 5.3. It means that the *read-pointer* points to the least significant occupied slot of buffer. At this point, the VC can only issues request to the arbiter leading its flits to exit the router without any new flit enters the VC. The *read-pointer* continues pointing to the occupied slots until it points to the least significant occupied slot of buffer again. At this point, the closing and requesting operations of the VC return to the normal condition.

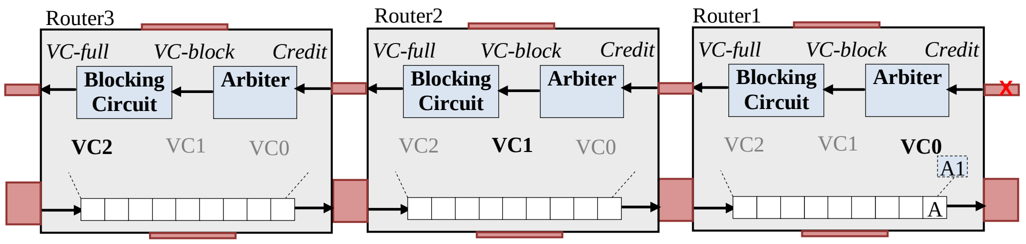
Table 5.1. Closing and Requesting Operations of RDQ Mechanism Associated with a VC

condition	*VC-block*	*Stop-Req*	*VC-req*	*VC-full*
1	0	0	x: Normal 1: *read-pointer* points to a flit of the VC. 0: r*ead-pointer* does not point to a flit of the VC.	x: Normal 1: Input-port buffer is full. 0: Input-port buffer is not full.
2	1	0	0: r*ead-pointer* points to no flit of the VC.	x: Normal 1: Input-port buffer is full. 0: Input-port buffer is not full.
		1	0 to1 then returns 0: *read-pointer* points to a flit of the VC.	1: stop incoming flit for the VC
3	1 to 0	1: *read-pointer registers' output* < input	0: stop request	1: stop incoming flit for the VC
		0: *read-pointer registers' output > input*	x: Normal	1: stop incoming flit for the VC
		0: *read-pointer registers' output > input*	x: Normal	x: Normal

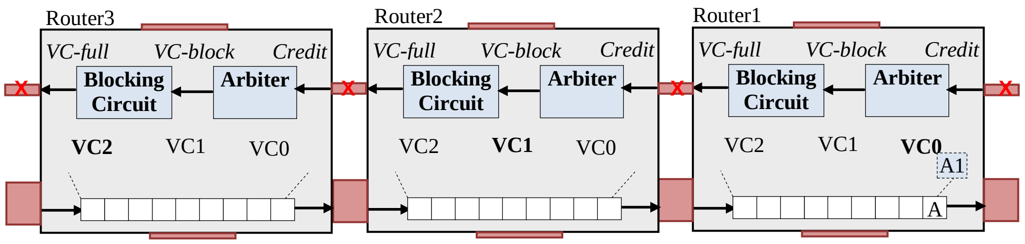
5.1.3 Back Pressure for Low NoC Traffic

A specific situation can arise in our RDQ mechanism when a packet blockage condition travels back to the up-stream routers that are related to the blocked packet. This situation assists the VC organization in maintaining the order of flits associated with the blocked packet. This also leads to chain-blocking and the creation of back-pressure in the NoC. The implementation of this situation has a direct affect on the performance of RDQ especially at low injection rates. The EDVC approach cannot handle this situation effectively [55] and when there is no credit for an output, the input VC associated with that output becomes blocked despite having some flit in the VCs related to blocked packet. Consequently, when there is no flits in all the upstream VCs related to blocked packet, the blockage spread back in the NoC.Even for low injection rate traffic, the blockage reaches to the source that will stop injecting any new flit. Moreover, the blockage removal at the upstream routers will also take more times as compared to downstream routers. In other words, a small blockage delay leads to a huge delay at the source core.

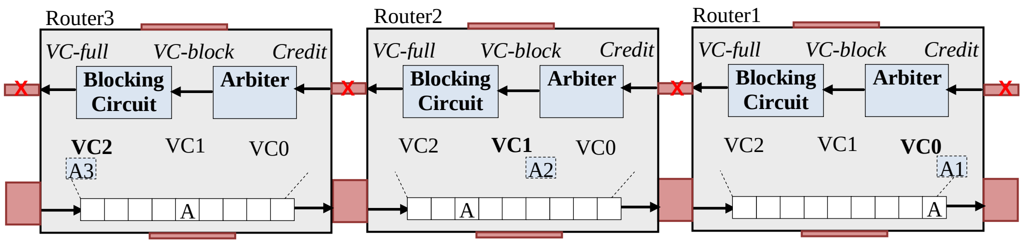
Consider the scenario of low flit injection rate shown in Figure 5.6, where the latency of the three mechanisms: EDVC, LLD and RDQ are illustrated. Assume that the flit injection rate is low e.g. 1/6 flit per clock cycle. The flits are injected by the source cores at 6 clock cycle intervals. Moreover, assume that flit arrival/departure delay of RDQ and EDVC routers are one clock cycle, and that of LLD router is 2 clock cycles.



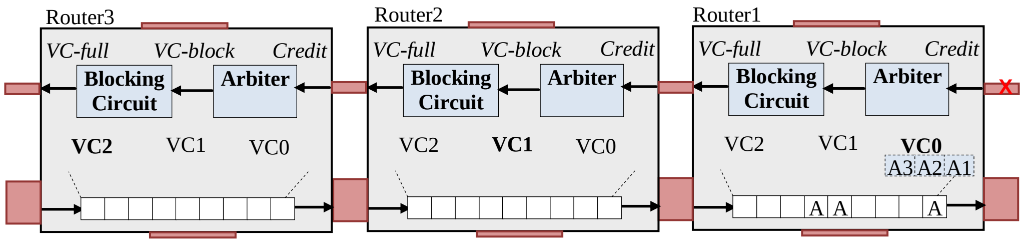
(a) EDVC: Packet *A* becomes blocked at the 3rd cycle,



(b) EDVC: All the related VCs are blocked at 18th cycle.



(c) RDQ: At least one flit saved in related blocked VCs at 18th cycle.



(d) LLD: Three flits are saved in the VC0 of router1 at 18th cycle.

Figure 5.6: Low Flit Injection Traffic Scenario in EDVC, LLD and RDQ

Assume at time zero, three flits *A1*, *A2* and *A3* are being injected with 1/6 injection rate, and these flits pass through three routers, *router3*, *router2* and *router1* reserving VC0, VC1 and VC2 of those routers respectively. Figure 5.6a demonstrates the EDVC mechanism such that the flit *A1* is blocked in VC0 of *router1* at the 3rd cycle (three cycles need to pass three routers). Then three cycles are needed for the blockage signal to go back and reach the source core located before *router3*. In the EDVC mechanism, the empty and reserved VCs can be blocked. Therefore, the second flit, *A2* cannot be injected, and this condition will remain till the blockage is removed at the 18th cycle as illustrated in Figure 5.6b. After 18th cycle, 7 clock cycles are needed to send out three flits *A1*, *A2* and *A3*. In other words, three cycles are needed for removing the blockage in *router1*, *router2* and *router3*, and four cycles are required for *A3* to pass through the three routers (including one clock to wait for the flit *A2)*. We assumed that the flits, *A2* and *A3* are injected (after 6 and 12 cycles delay) by the source core as soon as their outputs are free.

In the case of RDQ based VC-buffer organization, a VC can be blocked when at least one flit is saved in it as illustrated in Figure 5.6c. Flits *A1*, *A2* and *A3* are saved in *router1*, *router2* and *router3* by the 18th cycle. Then 5 clock cycles are needed to send out three flits from router1. In fact, it needs two cycles to remove the blockages and 3 cycles for passing the flit *A3* through three routers. In the case of conventional LLD, three flits are saved in the VC0 of *router1* by 18th cycle as shown in Figure 5.6d. There is no back-pressure, and we assume that VC0 capacity for *router1* is more than three flits. Then 6 clock cycles need to send out the three flits from *router1*. The packet flow cases of Figures 5.6a to 5.6d illustrate that the RDQ has a delay of 23 cycles, which is faster than the EDVC (25 cycles delay) and LLD (24 cycles delay) mechanism.

5.2 Index-based Round Robin Arbiter (IRR)

In this section, we present a new arbiter, Index-based Round Robin (IRR) arbiter that employs a least recently served priority scheme and achieve strong fairness arbitration. The proposed arbiter has smaller arbitration delay, lower chip area and it also consumes less power as compared to the arbiters described earlier in Section 2.6.2. Before describing the IRR arbiter architecture, we introduce an inseparable and critical output in the arbiter design.

5.2.1 Grant Index

All the arbiters have output signal, *grant* whose width is the same as that of input width. However, in practical designs, the index of *grant* signals, *g_id* is also generated that is used to address the granted request in some other components, such as control tables, multiplexers and memories used in NoC routers. When a crossbar switch is made of multiplexers, the *g_id* can be connected to the selection port of multiplexer to switch the granted input to the requested output-port (see Figure 2.23). The width of *g_id* is the $\log_2$ of the width of *grant*. We used the *g_id* as the first output of our proposed arbiter design and due to lower width of *g_id*, our arbiter design is smaller and faster as compared to other arbiters. Due to the critical use of *g_id* in NoC design, we consider all the arbiters covered in this chapter to generate both *grant* and *g_id* as outputs.

5.2.2 Fixed and Variable Priority Arbiter

Our fixed priority arbiter is simpler and economical and its details are illustrated in Figure 5.7. The priority of requests is linear and in the ascending order where r_0 has the highest priority. The index of first asserted request is switched to the output as the index of *grant*, *g_id*. Then the *g_id* is decoded to create the *grant* signals.

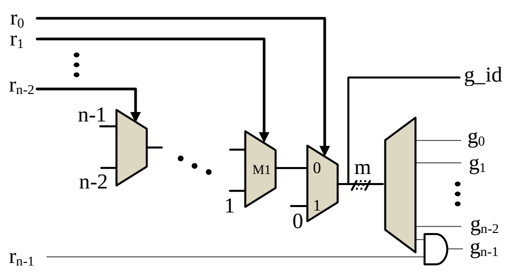


Figure 5.7: *n*-input fixed priority arbiter, where $m = \log_2(n)$.

The last request, r_{n-1} has a simplified circuit where instead of being multiplexed like other requests, it is ANDed by g_{n-1}. If the *g_id* output of the fixed priority arbiter of Figure 5.7 is connected to the last multiplexer, each request behaves as it has the highest priority through ascending order of the loop. For example, for four requests (r_0, r_1, r_2 and r_3) the output of multiplexer, *M1* generates an index where r_1 has the highest priority then r_2, r_3, and r_0. Therefore, by further multiplexing these outputs, we can choose an input as the highest priority request as shown in Figure 5.8. For example, when P=1, the output of multiplexer *M1* is selected and the request, r_1 has the highest priority. In the case of no request asserted i.e. r_0 is asserted, *g_id* issues the same value (i.e. zero). In order to separate these two conditions, ORing of requests, *any_r* is ANDed with the g_0 so that when all the requests are zero, all the *grant*s also become zero.

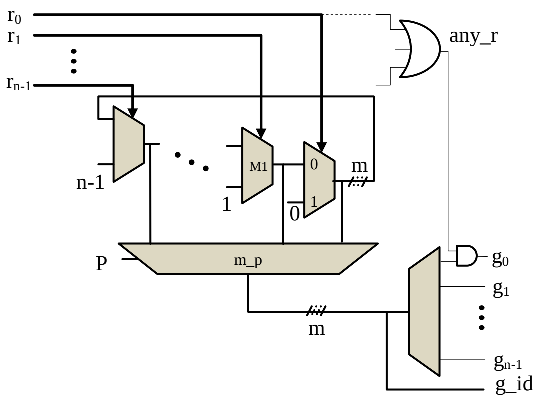


Figure 5.8: *n*-Input Variable Priority Arbiter

5.2.3 IRR Arbiter Micro-Architecture

If the next index of granted request is employed for the next priority selection, the current granted request receives the least priority, and its next request receives the highest priority among all the requests. To accomplish it, the *g_id* array is stored in a register whose output is incremented and connected to the selection port of multiplexer, *MP* as shown in Figure 5.9. In this way, the arbiter follows the least recently served priority scheme or a round robin scheme. Figure 5.9 illustrates our proposed IRR arbiter that takes one clock cycle for arbitration. To keep the priority unchanged, the priority register output, *next_g* is fed back into SF multiplexer to cater for no request. It guarantees strong fairness arbitration in our design discussed below.

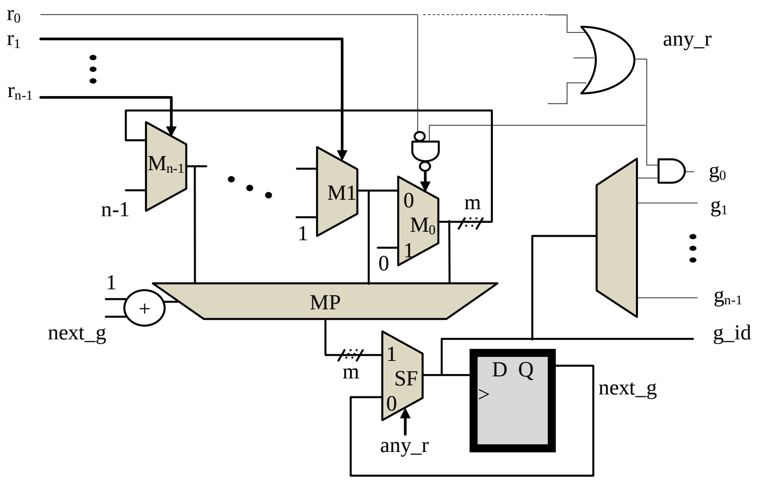


Figure 5.9: n-Input IRR Arbiter, where $m = log_2(n)$

5.2.4 Functional Behaviour of IRR Arbiter

We present the functionality and behaviour of our round robin arbiter illustrated by its timing diagram in Figure 5.10. During time 1-6, a fixed input request, ‘1111’ is applied and granted bit by bit per clock cycle. At time 6, the request is changed to ‘0000’, i.e. no request is asserted. For no request situation, the priority of last granted request is recorded and applied when a new request is asserted. For example, at time 6, the priority of second bit of the request is recorded and applied at time 8. Consequently, the forth bit of request is granted at time 8. We tested our arbiter along with some past arbiters (RoR, Matrix, PRRA, IPRRA, and HDRA) for the same testbench and request scenario, and timing results are shown in Figures 5.10 and 5.11. When no request is asserted, the RoR, Matrix and our IRR arbiters record the current priority shown in Figure 5.10. However, the PRRA, IPRRA and HDRA arbiters couldn’t record the priority and show different waveforms shown in Figure 5.11. For a no request condition, the highest priority is given to the least significant bit of the request for PRRA, IPRRA and HDRA waveforms. This arbitration behavior of PRRA, IPRRA and HDRA is due lack of any circuit to handle the no request condition. Keeping the last priority during no request condition has a direct impact on the fairness of an arbiter. The first advantage of our IRR arbiter is to perform a stronger fairness arbitration.

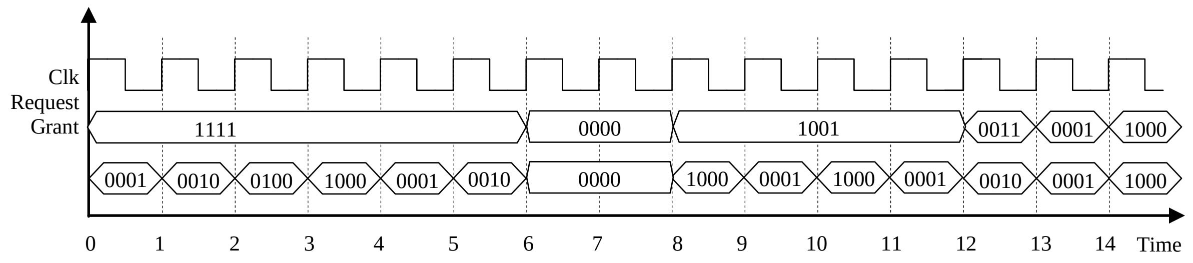


Figure 5.10: Timing diagram for some input request scenarios of strong fairness round robin arbiters.

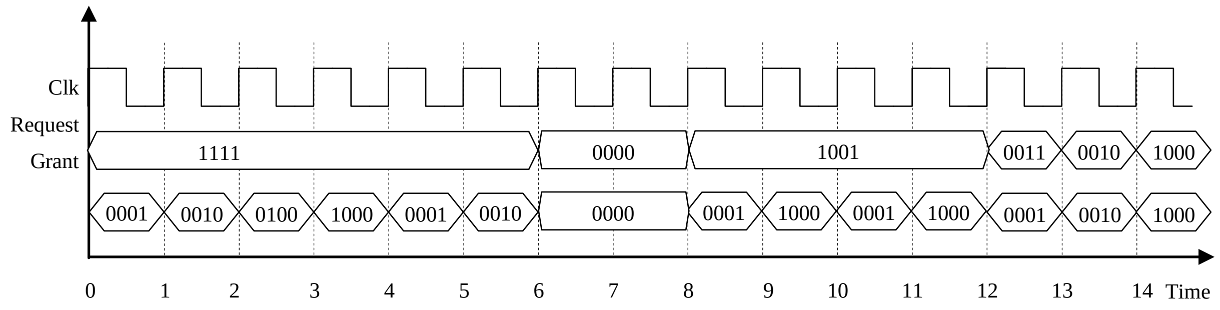


Figure 5.11: Timing diagram for some input request scenarios of weak fairness round robin arbiters.

5.2.5 IRR Hardware Analysis

We also perform a hardware overhead analysis to compare the expected speed and hardware overhead of the aforementioned round robin arbiters with our proposed IRR arbiter. We don't apply any algorithm to optimize the circuits as an Electronic Design Automation software does. The main figures of merit of an arbiter circuit are speed, area and power consumption. The usual measure for speed of an arbiter circuit is the delay time or maximum clock frequency (f_{max}). The clock frequency of an arbiter depends on the longest delay (critical path) between two registers clocked at the same time. The circuits of 4-input arbiters, which have been discussed in Chapter 2 (Figures 2.26, 2.27, 2.28, 2.29 and 2.30) and Figure 5.9, are decomposed at the gate level. The electrical parameters of the logic gates are derived from Synopsys 90nm Digital Standard Cell Library as listed in Table 5.2. We calculated the sum of the areas and powers of all the cells of each arbiter to estimate their power and area that are listed in Table 5.3. The power includes both static and dynamic powers. For speed estimation, the critical path delay between two registers of each circuit is calculated. We have also provided the critical path of each circuit through the number in parentheses for the last column of Table 5.3. The increment is done by a half adder (XOR2x1). As discussed earlier, RoR and Matrix arbiters have strong fairness, and the HDRA, PRRA and IPRRA are weak in fairness.

Table 5.2. Electrical Parameters Gates from Synopsys Library

Gate name	Propagation Delay (ps)	Static Power (nW)	Dy. Power (nW/MHz)	Area (um2)
INVX1	38	88	12	6.5
AND2X1	85	298	19	7.4
AND3X1	119	297	34	8.3
NAND2X1	51	336	15	5.5
OR2X1	85	226	23	7.4
OR3X1	114	250	39	9.2
OR4X1	137	261	56	10.1
NOR2X1	64	170	15	6.5
MUX21X1	107	815	43	11.1
MUX41X1	168	827	58	23.0
DEC24X1	119	1238	66	29.5
XOR2X1	133	454	26	13.8
DFFARX1	217	620	100	32.2

Therefore, we introduce a weak fairness version of IRR named IRR_WF for comparison with the HDRA, PRRA and IPRRA arbiters. The only difference between IRR and IRR_WF is the SF multiplexer shown in Figure 5.9 that affects the critical path delay. We expected a faster IRR_WF arbiter than IRR. Table 5.3 list the characteristics of all the arbiters. IRR has the optimum performance and hardware overhead among all the listed arbiters. In terms of speed, the IRR can run 15% to 50% faster than the other arbiters. Moreover, the IRR saves the chip area from 10% to 47% and power from 1% to 44%. We also evaluated our IRR arbiter with the other arbiters in terms of area, power and timing by using the EDA tool in the experimental section of this chapter.

Table 5.3. Characteristics of 4-input Arbiters based on Table 5.2

Type of 4-input arbiters	Area (um^2)	Power (uW)	Critical path Delay (ps)
IRR	294	296(282[d])	625 (217+133+168+107)
RoR	328	298(289[d])	1242(217+5*(85+85)+137+38)
Matrix	556	479(465[d])	747 (217 +2*38 +3*85+114+85)
IRR_WF	280	274(262[d])	518 (217+133+168)
HDRA	431	360(348[d])	609 (217 +64+51+85+85+107)
PRRA	510	493(479[d])	861(217+2*38+3*85+85+2*114)
IPRRA	528	488(473[d])	747 (217+2*38 +3*85+85+114)
IRR/RoR	10% Saving	1% Saving	50% Faster
IRR/Matrix	47% Saving	38% Saving	16% Faster
IRR_WF/HDRA	35% Saving	24 % Saving	15% Faster
IRR_WF/PRRA	45% Saving	44% Saving	40% Faster
IRR_WF/IPRRA	47% Saving	44% Saving	31% Faster

[d] dynamic power

5.3 Switch Allocator

While an arbiter arbitrates among multiples requesters for a single resource (e.g. output-port), an allocator arbitrates among a group of requesters for a group of resources. Each requester may request one or more of the resources, but each resource is assigned to only one requester [22, 56]. An allocator structure is utilized in the Switch Allocator (SA) module of the arbitration module in NoC router, which will be described in this chapter. The allocators can be designed in a unit form such that the requesters are interconnected in order to maximize the matching among the requesters and the resources. The allocators in the form of unit design are difficult to be parallelized or pipelined, and they are too slow for applications in which latency is important [22]. Latency-sensitive applications typically employ fast and fair matching designs such as separable allocators. A separable allocator has significantly less logic complexity by separating the requesters in some groups. Moreover, the critical path delay of a separable allocator can be improved by choosing a fair and fast arbitration among each group. For these reasons, separable allocators are usually utilized and investigated in most of the NoC research projects. In a separable allocator, two sets of arbiters can perform arbitration: one across the inputs and one across the outputs as illustrated in Figure 5.12. In an input-first separable allocator, arbitration is first performed to select a single request at each input-port. Then, the outputs of these input arbiters become the inputs to a set of output arbiters to select a single request for each output-port [22].

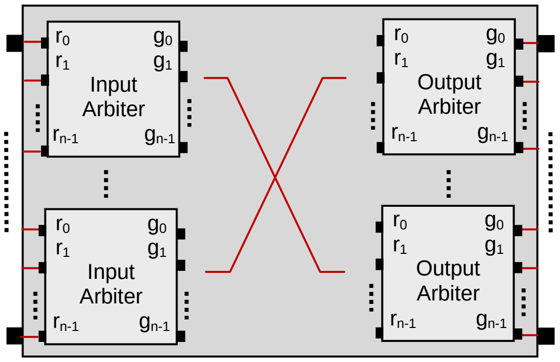


Figure 5.12: An Input-First Separable Allocator.

In order to ensure fairness, avoid traffic starvation, and to perform arbitration in a single iteration, a RR scheme can be utilized for any given arbiter which is shown in Figure 5.12.

5.3.1 NoC Switch Allocator Function

Figure 5.13 shows an input-first separable SA (switch allocator) micro-architecture implemented in our design. The first set of modules, *input-arbiter*s perform arbitration among the VCs of each input-port of the router. Therefore, the size of each *input-arbiter* is $v \times v$, where v is the number of VCs in each input-port (v=4 in the architecture of Figure 5.13). The number of *input-arbiter*s, n is the same as the number of input-ports in a router (n= 5 in the architecture of Figure 5.13). The *decoder* modules generate the requested outputs of the winner VCs of input-ports as illustrated in the lower part of Figure 5.13. Each bit of *decoder* output is corresponding to an output-port, and each active bit of *decoder* output shows the requested output by the winner VC of relevant input. Assume that the *v0* becomes winner in the *in_3* arbiter, whereas it requests the second output-port. Therefore, the second bit of the *c_3* output i.e. *out1* becomes asserted, and the remaining bits are de-asserted. The second set of arbiters, *output-arbiter*s performs arbitration among the winner input-ports for the output-ports. Therefore, the size of each *output-arbiter* is $n \times n$. The number of *output-arbiter*s is *m*, where *m* is the number of output-ports in the router ($n=m=5$ in the architecture of Figure 5.13). Assume the inputs, *In0, In2, In3* and *In4* make request for *out_1*. When *In3* wins, bit *g_out1_in3* of *out_1* arbiter is asserted, and rest of the bits are de-asserted. It should be considered that the winner VC of *In3* (assume *v0*) is already determined in *In_3*.

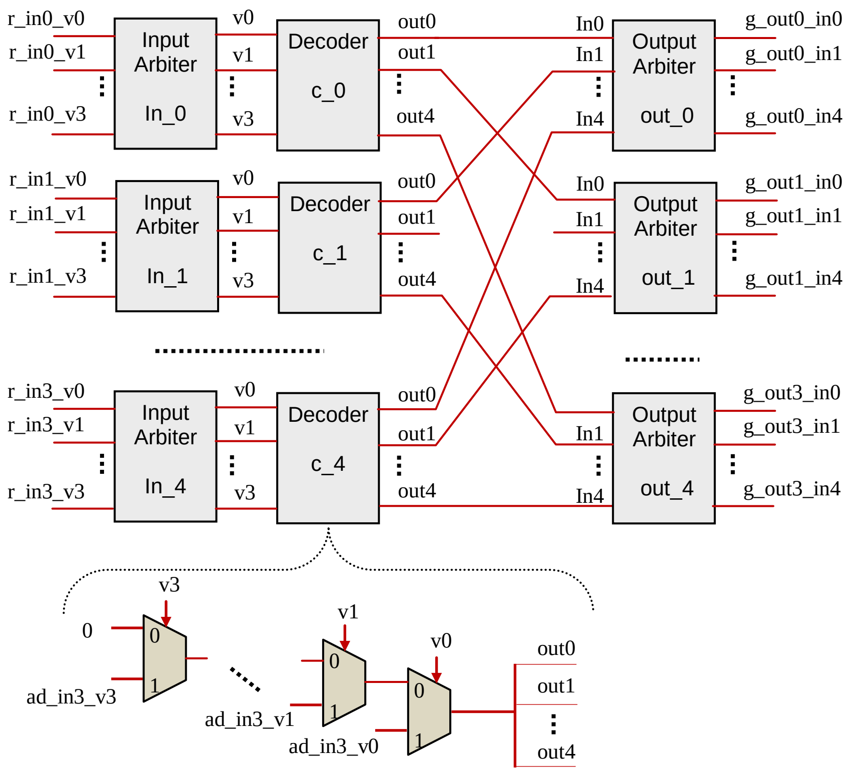


Figure 5.13: A 5×5 RR-Based Separable SA Micro-Architecture

5.3.2 VC Arbitration

In the previous section, we have discussed that the input-port VC arbitrations are usually implemented in SA stage by means of *input-arbiter* modules as illustrated in Figure 5.13. However, in our NoC arbiter design they are implemented in input-port by means of *VC-Selector* modules due to two reasons. Firstly, a central buffer stores all the VC flits of an input-port. Secondly, the arbitration is done in one clock event. In fact, when a *grant* signal is issued to an input-port, the *read-pointer* should have been already pointing to winner VC flit, or the winner flit should be loaded at the output-port of the buffer. The input-port micro-architectures of Figure 2.11 and 5.1 illustrate this scheme in such a way that the *VC-Selector* selects a VC for requesting to the arbiter, and simultaneously the flit of the VC is loaded at the buffer output. Moreover, the output of *VC-Selector* is used in the input-port mechanism. The *VC_req* signals in the LLD input-port of Figure 2.11 are used to generate *read-pointer*, and for RDQ input-port of Figure 5.1 these signals are used in the blocking mechanism. One may think that the *VC-Selector* modules can be accommodated in the switch allocator, and the *VC_req* signals can be fed back to the input-port. The problem of this design is that the input-port and arbiter become dependent on each other in terms of hardware and their speed. In fact, part of the critical paths of input-ports is shared with the arbiter. To prevent of such sharing, we implement the input-port VC arbitration as part of the input-port, and remove the *input-arbiter* modules from SA as illustrated in Figure 5.14.

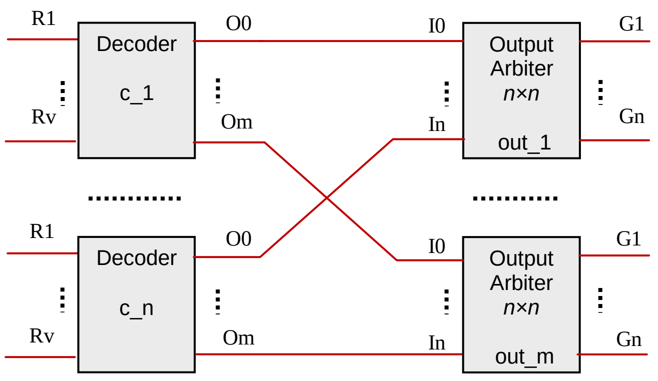


Figure 5.14: $n\times m$ SA architectures, n= # of inputs, m= # of outputs, v=# of VCs per input-port.

5.3.3 Post Switch Allocator Circuits

The micro-architecture of the Selection module is presented in Figure 5.15 that generates the credit and selection address of crossbar multiplexers (see Figure 1.5) related to an output-port of the router. When *g_in3_out1* is asserted, the circuit generates the number 3 at the *Sel* output that leads the input-port 3 to be connected to output-port 1 in the crossbar switch (see Figure 1.5).

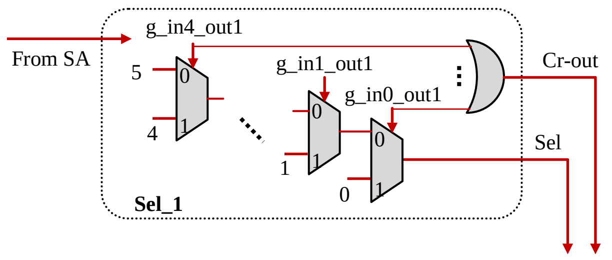


Figure 5.15: Output-Port 1 of Selection Module

Moreover, a credit signal, *cr_out* is issued to the Switch Traversal (ST) module. The ST module issues a credit, *credit-out* signal to the 1st output-port to store the flit in the associated downstream VC at the following clock events. Figure 5.16 shows the Grant module that generates the *grant* signal associated with the 3rd input-port of the router. The asserted *g_in3_out1* allows the flit of VC0 to transmit across the crossbar switch.

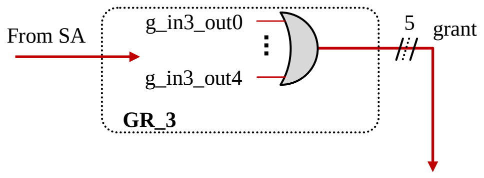


Figure 5.16: Input 3 of the Grant Module

5.3.4 FIFO Arbitration and VC Selector

We discussed the selection of a VC from the input-port VCs in LLD and ViChaR in Chapter 2. It is implemented in the *VC-Selector* module by employing a fixed priority arbiter as illustrated in Figure 2.16. The fixed priority arbiter is substituted with a Round Robin (RR) arbiter to ensure fairness and avoid traffic starvation as shown in Figure 5.17.

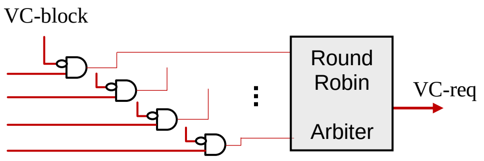


Figure 5.17: RR-based *VC-Selector* Utilized in LLD and ViChaR.

However, selection of a VC from the input-port VCs in RDQ mechanism happens in the data flow mechanism as discussed in Section 4.1.1. It is based on the location of *read-pointer* and the state of data in the input-port as indicated in Figure 5.18. In fact, the RDQ-based *VC-Selector* is part of input-port mechanism, and it cannot be substituted with a RR arbiter.

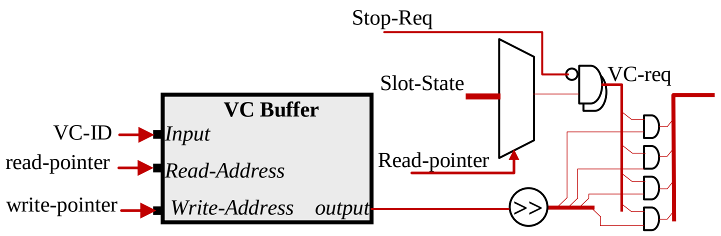


Figure 5.18: RDQ-Based *VC-Selector* in the Input-port Mechanism.

The *VC-req* is updated according to the location of the flit data in the input-port buffer. This feature of RDQ *VC-Selector* requires the arbitration among the input-port VCs to follow a FIFO fairness priority [42]. In FIFO fairness priority, the requesters are granted in a FIFO order of their requests.

Consider a scenario of data flow in an input-port as illustrated in Figure 5.19. There are 4 VCs containing data flits. In LLD and ViChaR mechanisms, all the four VCs issue requests to *VC-Selector*. The VCs are sequentially selected due to RR priority employed in *VC-Selector*. For example, in Figure 5.19 VC0, VC1, VC2, VC3, VC0, VC1, VC2 and VC3 and their associated flits, H0, B1, B2, H3, B0, T1, B2 and B3 can become winner in the *VC-Selector* sequentially. On the other hand, the VCs in RDQ mechanism are selected according to the location of *read-pointer* and the position of data flits in the input-port buffer. Assume that the *read-pointer* is located at the rightmost side of buffer in Figure 5.19 and counts toward the left side of buffer, and a flit in the right side is stored sooner than a flit in left of the buffer. The VCs, i.e. VC0, VC0, VC0, VC1, VC0, VC3, VC2 and VC2 and their associated flits, H0, B0, B0, B1, T0, H3, B2 and B2 become winner in *VC-Selector* sequentially. The order of VC selection in this scenario follows the order of stored flits, which is a FIFO order. The *write-pointer* also counts from right to left. In this way, as *read-pointer* and *write-pointer* always count in a direction, the *VC-Selector* in RDQ router follows a FIFO fairness priority scheme. However, in the case of a blockage, the flits associated with a blocked VC remain in the buffer until the blockage is removed. When the blockage is removed, as the order of stored flits is not in a FIFO order, the flits of blocked VCs may be serviced late according to the location of *read-pointer*. The fairness of VC arbitration in a RDQ router is as strong as those of LLD and ViChaR routers, and the arbitration avoids traffic starvation.

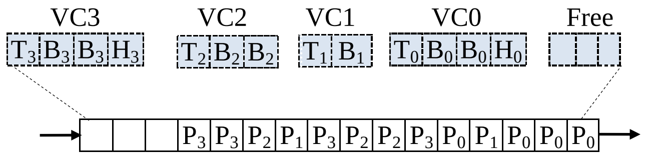


Figure 5.19: VC0, VC1, VC2 and VC3 Arbitration. RR for LLD and ViChaR and FIFO Priority in RDQ Routers.

5.3.5 RDQ Router Arbitration

In conventional DAMQ architectures (such as LLD), when the requested output of a VC is blocked (i.e. no output credit), the arbiter issues a block signal that causes the input-port to select any other available VC for service. No output credit means the corresponding *VC-full* of the downstream router is set. However, the RDQ arbiter will issue the block signal under two conditions i.e. either on losing switch arbitration to some other input-port or no output credit. This behavior of RDQ approach requires some extra hardware in the arbiter of the router as compared to LLD. The LLD arbiter updates a table at each clock cycle. The table consists of an array of registers where each bit represents the blocking state of a VC of the input-ports. Figure 5.20a illustrates the LLD blocking circuit of one-bit register associated with VC1 of input-port 3. If there is no output credit at each clock cycle, the *VC-block* signal is asserted, otherwise it is de-asserted. In our RDQ approach, the blocking circuit consumes extra hardware (an OR gate per VC) as shown in Figure 5.20b. When the requested output is closed or the input VC loses arbitration to other input-ports, the *VC-block* is asserted; otherwise, it is de-asserted.

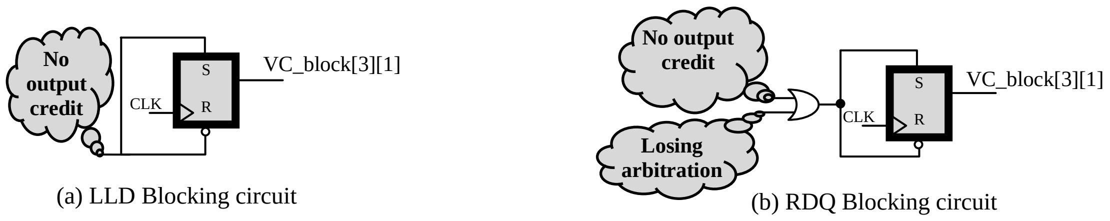


Figure 5.20: VC-Block Circuit Associated to VC1 of Input-Port 3

5.4 RDQ Based Router Architecture

In this section, we present the micro-architecture of RDQ router. First of all, we investigate the effect of RDQ port and IRR arbiter presented in this chapter on the efficiency of NoC systems. As already discussed, one of the important contributions of our approaches is to provide a simpler NoC router architecture. This architectural simplicity of our approach also leads to higher speed (higher clock rates) of NoC circuitry that will be discussed in the following sub-sections, while presenting our RDQ router architecture.

5.4.1 Rapid NoC Circuit Design

NoC architectures have been commonly presented in Globally Asynchronous Locally Asynchronous (GALS) design style [23], and we have also followed the GALS style in our router designs for 2D mesh NoC. In NoC GALS designs, the routers are locally synchronous (thus they are easier to be designed), but the NoC architectures are globally asynchronous, i.e. there can be varied clock rates for routers. In other words, the routers are independent in terms of clock design, and the faster clock rates of routers leads to the faster speed of NoC. NoC architecture includes a network of switches (routers) that are interconnected by data links as illustrated in Figure 1.1a. The structures of data links can be either simple wires or complex communication mechanisms like FIFOs. The data links in our design are considered as wires, and we assume that they do not affect the speed or performance of NoC. Regarding the NoC routers, we already described their structures in Chapter 1 and mentioned that the NoC routers investigated in this thesis consist of some input-ports, an arbiter and a crossbar switch as illustrated in Figure 5.21 and Figure 5.22. The NoC router illustrated in Figure 5.21 utilizes the LLD input-port and HDRA arbiter related to the past approaches. The router is called LLD-HDRA. The two proposed components i.e. the RDQ input-port and IRR arbiter are utilized in our NoC router that is called RDQ-IRR and illustrated in Figure 5.22. Except the Routing Computation (RC), Virtual channel Allocation (VA), and Switch Traversal (ST) modules, the other components of the routers shown in Figures 5.21 and 5.22 have been discussed in detail in the earlier sections of this chapter. In the following sections, we discuss the structures of RC and VA router components that determine the speed of router and consequently the NoC.

5.4.2 Fast Router Circuits

The digital circuits of a router can be divided into synchronous and asynchronous components. In asynchronous components, the state of the components can change at any time in response to their changing inputs. Therefore, the speeds of these circuits are almost proportional to the ASIC technology of silicon. For example, an asynchronous circuit created with 15nm IBM technology will run faster than that of 32nm IBM technology. The state of synchronous components changes only in response to their clock signals. Therefore, the speeds of these circuits are determined based on their maximum possible clock rates (f_{max}). The f_{max} is determined by critical path, i.e. the slowest logic path in the circuit. In an NoC router consisting of some pipelined components, the synchronous components determine the speed of the router. The synchronous components of a NoC router are those that utilize synchronous data buffers (e.g. registers or RAM). The crossbar switch component has an asynchronous architecture in our design, and it does not affect the speed of router.

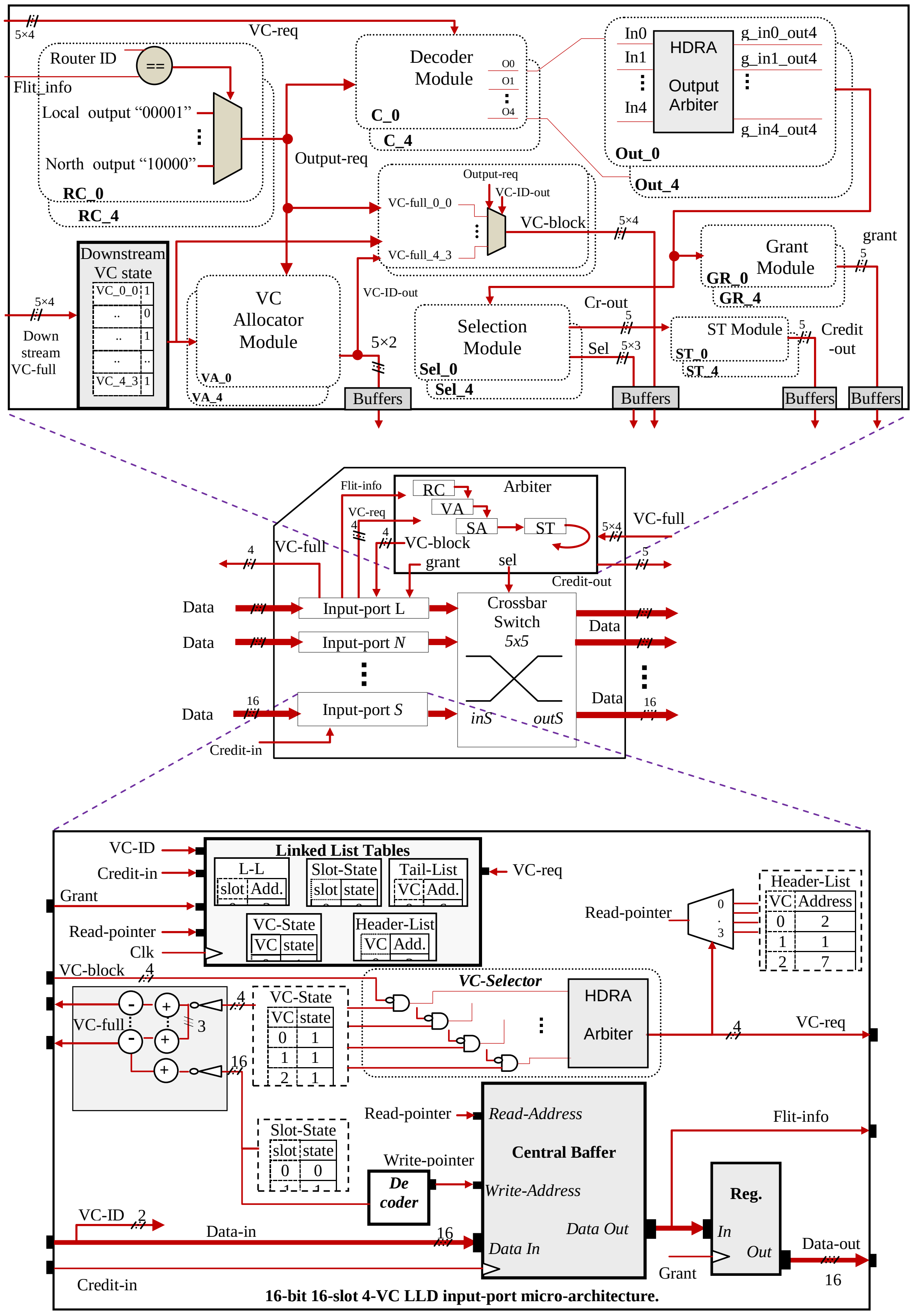


Figure 5.21: A LLD-HDRA Router Micro-Architecture.

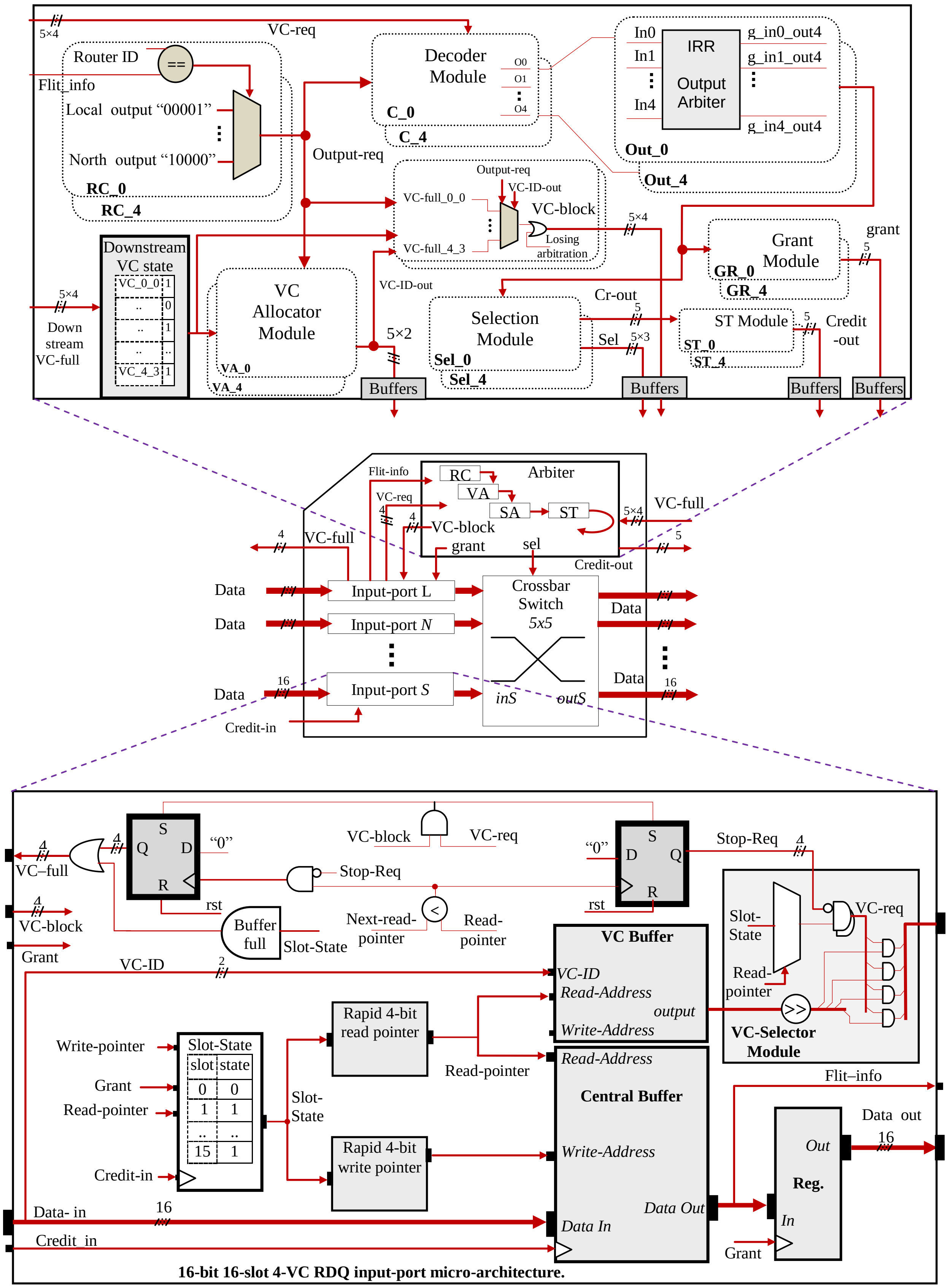


Figure 5.22: A RDQ-IRR Router Micro-Architecture.

However, two components i.e. input-port and arbiter utilize synchronous buffers to temporarily store data flits or information and affect the speed of a router. These two components are investigated in terms of their pipelined stages as part of our research.

5.4.3 Pipelined RDQ-based Routers

The RDQ port model that is an updated and faster version of the EDVC port behaves like a static model as discussed in Chapter 4 for the EDVC router. Therefore, the RDQ-based router pipeline consumes two clock events in a squeezed scheme if the arbitration takes one clock event (step), as illustrated in Figure 5.23a. We have also explained that the LLD and ViChaR port models are table based where their associated router pipeline consumes four clock events in a squeezed design if the arbitration takes one clock event (step), as shown in Figure 5.23b. In this way, the pipeline stages in LLD and ViChaR routers take two clock events longer than that of RDQ router, where arbitration is assumed to take one clock event.

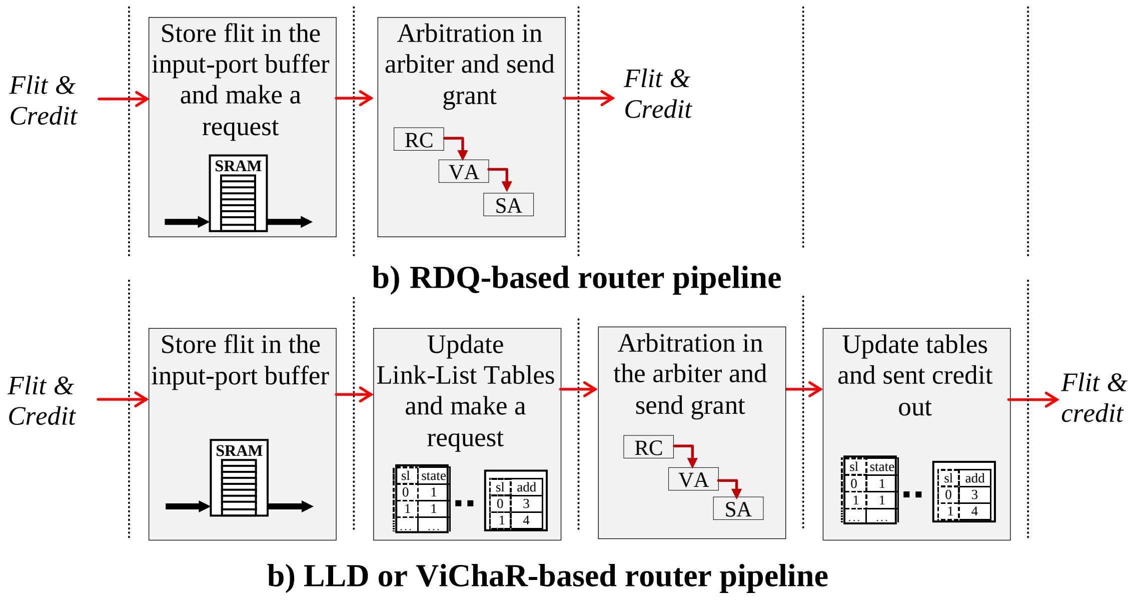

Flit & Credit
Store flit in the input-port buffer and make a request
SRAM
Arbitration in arbiter and send grant
RC
VA
SA
Flit & Credit
b) RDQ-based router pipeline
Flit & Credit
Store flit in the input-port buffer
SRAM
Update Link-List Tables and make a request
sl state 0 1 1 1
sl add 0 3 1 4
Arbitration in the arbiter and send grant
RC
VA
SA
Update tables and sent credit out
sl state 0 1 1 1
sl add 0 3 1 4
Flit & credit
b) LLD or ViChaR-based router pipeline

Figure 5.23: RDL vs. Conventional Dynamic Input-Port (LLD or ViChaR) Router Pipelines

The arbitration stages in our implementation follow the timing diagram of Figure 1.13. Each head flit of a packet must proceed through the stages of Routing Computation (RC), Virtual channel Allocation (VA), Switch Allocation (SA), and Switch Traversal (ST). To better illustrate the pipelined stages of our arbiter, the micro-architectures of the aforementioned stages are described in the following section.

5.4.4 Pipeline Stages Micro-Architectures

The structure of RC can be a simple multiplexer as shown in Figure 5.24, because a simple XY routing algorithm is considered for communication in our 2D mesh NoC design. The figure illustrates that the ID of destination, *Flit_info* (which is already stored in the header flit of a packet) is compared with the ID of the current router. Then, the requested output-port is generated at the RC output according to XY routing algorithm.

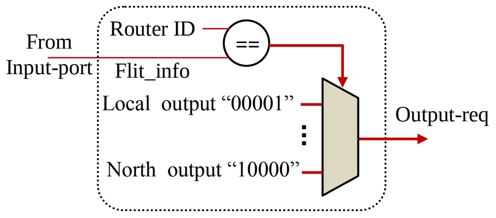


Figure 5.24: RC generates free requested output VC

The structure of VA can be a fixed-priority arbiter as given in Figure 5.25. The first free VC of the downstream router input-port is selected in an ascending order. The fixed-priority arbiters are simple and have been discussed in detail in Section 2.6.1. The SA module includes the Decoder, Output-arbiter, Selection and Grant modules that have been presented in Section 5.3.

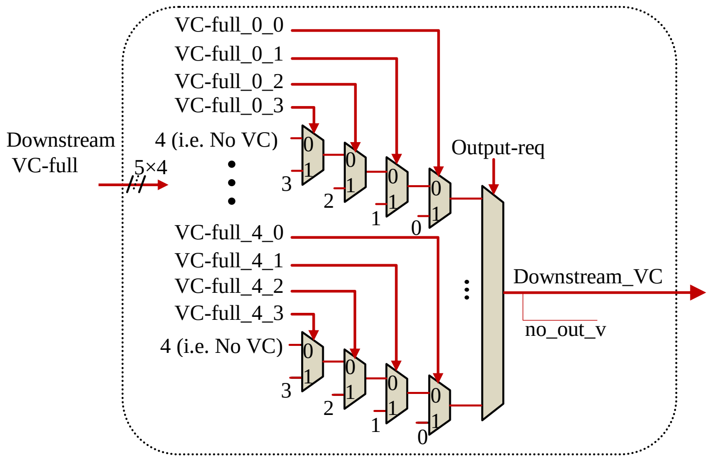


Figure 5.25: VC Allocator generates free downstream VC

The structures of arbiter sub-modules of routers as given in Figures 5.21 and 5.22 such as RC (see Figure 5.24), VA (see Figure 5.25), Decoder (see downside of Figure 5.13), *output-arbiter* (see Figures 2.26, 2.27, 2.28, 2.29, 2.30 and 5.9), Selection (see Figure 5.15) and Grant (see

Figure 5.16) illustrate that the circuit routes from the inputs of RA to the outputs of Selection, Grant or VA modules are not sequential. In other words, the outputs of Selection, Grant or VA modules are not relative to clock cycle. Therefore, the RC, VA and SA stages can determine the winner VC and winner input-port of each output-port in a single iteration. In this way, the data flit related to winner VCs can get out of the router at the following clock event, and the VCs can make new requests. There is no intermediate register among the paths from the inputs of RC to the outputs of SA, and the registers related to the *output-arbiter*s only keep the state of SA for the following clock event (Figure 5.9 and 5.14). Among the longest path that passes through the RC (Figure 5.24), VA (Figure 5.25) and SA (Figure 5.14, 5.15 and 5.16), there is no intermediate registers. In this way, all the arbitration stages are performed in one clock event.

At the end of SA stage, if the *output-arbiter* output associated with a VC is active, the following actions are performed according to the kind of flits.

- If the request is from a header flit, the associated RC and VA outputs are reserved, and the associated downstream router VC, *grant*, and crossbar address are issued.
- If the request is from a body flit, the associated downstream router VC, *grant*, and crossbar address are issued.
- If the request is from a tail flit, the reservations are removed, and the associated downstream router VC, *grant*, and crossbar address are issued.

The credit, *credit_out* signals associated with *grant*s are issued at ST stage to store the data in the downstream input-port at the following clock event. In fact, the signals, *cr_out* generated in Selection module of Figure 5.15 are repeated in the ST module at the following clock event. In the following sections, we experiment the above features and advantages of our approaches, especially their effects on the performance of NoC system are investigated.

5.4.5 Communication in RDQ-IRR router

In this section, RDQ-IRR data flow mechanism is presented when there is no contention in the NoC communication and the buffer is empty. We employ asynchronous communication among the routers. The following steps describe the working of RDQ-IRR as illustrated in Figure 5.22 according to the timing diagram given in Figure 5.26.

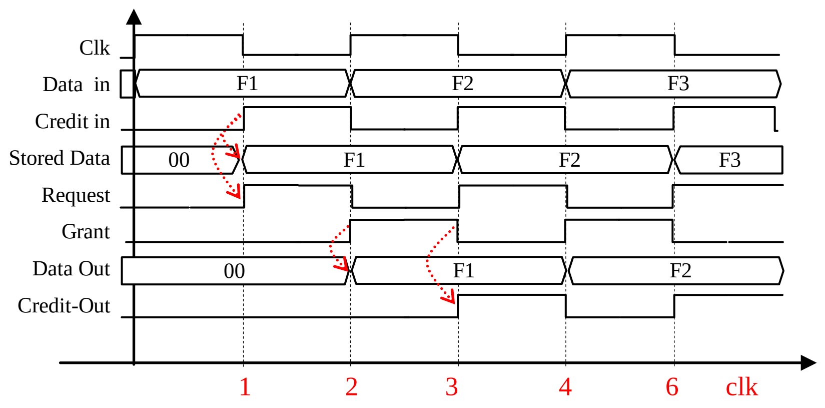


Figure 5.26: Three Steps of RDQ-IRR VC Flow Control.

Flit Arrival State (e.g. Clock edge #1 of Figure 5.26)

1) A *Credit-in* signal causes the incoming flit (e.g. flit F1) and its *VC-ID* to be saved in a slot pointed by the *write-pointer*. Meanwhile, the corresponding bit of the *Slot-State* table is set.
2) When the *read-pointer* points to a slot and its *Slot-State* bit is set (occupied slot), a request signal is issued by the *VC-Selector* module according to the *VC-ID*. The *read-pointer* also causes the flit to appear at the output of central buffer that leads the flit information (*flit-info* signals) to be read by the arbiter.
3) The RC module of the arbiter will read the flit information and determine its requested output-port.
4) The VA module reads the RC output and state of VCs in the downstream router, and determines if a free VC of the downstream router input-port is available.
5) If the VC is available, then the SA module (Decoder and Output-Arbiter modules) reads the RC output and *VC-req* signals and performs arbitration among the winner VCs for the output-ports. (see Sections 5.3.1and 5.3.2)
6) The *Selection* module reads the SA output and determines the associated address for the crossbar switch module.
7) The *Grant* module reads the SA output and sets the grant signals for the winner VCs.
8) The *VC-block* module generates the *VC-block* signals under two conditions.
    - Losing switch arbitration to some other input-ports. It happens when *VC-req* is set, but its associated *grant* is reset.

- No output credit (see Section 5.3.5). It is determined by RC, VA and *Downstream-VC-state* outputs as illustrated in the *VC-block* module of Figure 5.22.

9) The ST module reads the *Cr-out* signals and keeps record of them to issue *credit-out* signals after two clock events.

Flit Departure State (e.g. Clock edge #2 of Figure 5.26)

All the signals produced in steps 2 to 8 are issued to their associated modules as described below.

10) A *grant* signal causes the associated flit (e.g. flit F1) to exit the input-port and the corresponding bit of the *Slot-State* table is reset.
11) The *Sel* signals cause the crossbar module to switch the input-ports to their associated output-ports.
12) The *VC-ID-out* signals carry the *VC-ID* of the flit.
13) In case of no output credit or arbitration loss, the *VC-block* signals cause the associated VC to become blocked (there is no *grant* signal).

Credit and Next Flit Arrival State (e.g. Clock edge #3 of Figure 5.26)

14) The *credit-out* signals are issued by the ST module and cause the transferred flits (e.g. flit F1) to be stored in the downstream router input-ports' buffers.
15) Steps 1 to 9 are repeated for the next incoming flits (e.g. flit F2, F3, etc.).

5.5 RDQ and IRR based NoC Experimental Results

Our novel RDQ and IRR organizations and structures presented in this dissertation are modelled and experimented in three parts here. First of all, we evaluate the IRR arbiter as compared to some previous arbiter designs in terms of its speed and hardware overhead. The second part of our experiment is related with RDQ based input-port organization. RDQ input-port is investigated and evaluated as compared to some previous input-port designs in terms of performance and hardware requirements. Finally, the overall effects of employing IRR and RDQ input-port are investigated on the performance, hardware and speed of NoC systems.

5.5.1 IRR Arbiter Evaluation

IRR arbiter is evaluated and compared with other arbiters. To analyze the speed, area and power overhead, all the arbiters are implemented in System Verilog and synthesized using the

Synopsys Design Compiler for 90nm Synopsys Generic Library and Altera FPGAs (Stratix V). The resulting designs operate at 400 MHz and 1.2 Volt. Different structures of each arbiter are synthesized, and their results for total chip area, critical path delay, and power (dynamic as well as static) consumption are listed in Tables 5.4 and 5.5. The Synopsys Design Compiler executes various algorithms iteratively to find an optimum architecture. There is always a trade off among the power, area and critical path characteristics of a design. This trade off behaviour has dictated us to consider the arbiter structure that has smaller power, area and critical path delay as an efficient design. The critical path is the limiting factor preventing us from decreasing the clock period. For a fair comparison, we group the arbiters into strong and weak fairness arbiters and present their results in Tables 5.4 and 5.5. Table 5.4 lists the IRR, RoR and Matrix arbiter characteristics. We listed two synthesised versions of IRR at 16 and 32-input configurations indicating the IRR having efficient area, power and timing characteristics than RoR and Matrix arbiters.

Table 5.4. Hardware Characteristics of Strong Fairness Round Robin Arbiters

Input	Design	ASIC design (90 nm Generic Library)			FPGA design (Stratix V)	
		Total Cell Area *(μm^2)*	Power *(μW)*[a]	Critical path*(ns)*	Comb. Element	Reg. *(bits)*
4	RoR	299	76	0.99	13	4
	IRR	295	53	0.54	9	2
	Matrix	431	80	0.56	16	6
8	RoR	1066	204	0.90	25	8
	IRR	705	105	0.66	37	3
	Matrix	1838	313	0.73	58	28
16	RoR	1846	251	1.46	56	16
	IRR	1390	155	1.24	95	4
		2140[b]	182[b]	0.71[b]		
	Matrix	7817	1242	0.73	238	120
32	RoR	3175	384	2.39	109	32
	IRR	2859	219	1.23	205	5
		4224[b]	351[b]	0.89[b]		
	Matrix	31498	4663	0.92	958	496
IRR/RoR		19% saving	44% saving	43% shorter	28% extra	73% saving
IRR/Matrix		70% saving	75% saving	7% shorter	48% saving	90% saving

[a] Frequency for power estimation= 400 MHz.
[b] The second version of IRR for comparison with the Matrix arbiter.

It consumes on average 19% and 70% less chip area, 44% and 74% less power (static and dynamic), and 43% and 7% shorter critical path delay as compared to RoR and Matrix arbiters respectively. The results of 4-input arbiters follow the analytical results of Table 5.3. We consider IRR_WF for comparison with the HDRA, PRRA and IPRRA arbiters. Table 5.5 shows the analysis results indicating that IRR_WF is more efficient as compared to PRRA and IPRRA arbiters for all the input configurations.

Table 5.5. Hardware Characteristics of Weak Fairness Round Robin Arbiters

Input	Design	ASIC design (90 nm Generic Library)			FPGA design (Stratix V)	
		Total Cell Area *(μm^2)*	Power *(μW)*[a]	Critical path delay *(ns)*	Comb. element	Registers *(bits)*
4	IRR_WF	278	45	0.48	9	2
	HDRA	352	117	0.6	11	4
	PRRA	344	85	0.79	9	4
	IPRRA	373	74	0.69	9	4
8	IRR_WF	667	100	0.61	37	3
	HDRA	754	172	0.72	28	8
	PRRA	777	158	1.02	24	8
	IPRRA	848	153	0.87	28	8
16	IRR_WF	1478	179	0.77	95	4
	HDRA	1588	249	0.85	62	16
	PRRA	1683	262	1.25	56	16
	IPRRA	1809	267	1.05	73	16
32	IRR_WF	2801	211	1.11	205	5
		3831[b]	315[b]	0.93[b]		
	HDRA	3204	398	0.94	125	32
	PRRA	3466	442	1.48	126	32
	IPRRA	3719	454	1.27	159	32
IRR_WF/HDRA at 4, 8 and 16-inp.		Saving 13%	Saving 44%	Shorter 15%	Extra 12%	Saving 63%
IRR_WF/PRRA		16%	46%	34%	33%	73%
IRR_WF/IPRRA		22%	45%	24%	23%	73%

[a] Frequency for power estimation= 400 MHz.

[b] IRR_WF for comparison with the HDRA arbiter.

It saves on average 16% and 22% less chip area, 46% and 45% less power, and 34% and 24% shorter critical path delay as compared to PRRA and IPRRA arbiters. The IRR_WF is also more efficient as compared to HDRA arbiter for various input arbiter structures. It saves 13% less chip area, 44% less power, and 15% shorter critical path delay on average. Overall, IRR and IRR_WF arbiters consume the least amount of power among all the arbiters due to its usage of fewer registers. The fewer number of registers wed in IRR also leads to simpler design and chip layout due to a simple clock tree organization. To illustrate the clock tree advantage, we measured the maximum frequency, f_{max} of IRR_WF and HDRA for (32-input configuration) on FPGA implementation. The IRR_WF arbiter can operate at 15% higher frequency as compared to HDRA when implemented on an FPGA.

5.5.2 RDQ based Input-Port Implementation and Results

Hardware characteristic of RDQ based router input-port is compared with three DAMQ based input-port and VC organization mechanisms commonly known as LLD [14], ViChaR [13] and EDVC [55]. We have implemented the input-port micro-architectures using SystemVerilog. The hardware parameters are determined by employing Synopsys Design Compiler for generic 32nm NAND technologies. Also, some parameters are derived from Quartus-II for Stratix-V FPGA. The setup constrains as well as CMOS technology parameters of global operating voltage of 0.85V and time period of 2.5*ns* (400MHz) is applied for input-port design of all the four mechanisms including LLD, ViChaR, EDVC and RDQ. The width of slot buffer is set to the flit size i.e. 16-bits. The resulting characteristics of input-port micro-architecture are listed in Table 5.6 for different buffer size in terms of slots. The RDQ input-port has the optimum chip area, power and timing characteristics among all the other input-ports. On average, the RDQ input-port consumes 13% less IC area, 21% less power and has 49% less critical path delay for an ASIC design. It also has 10% less registers for an FPGA design as compared to LLD based input-port. In terms of FPGA combinational elements, the RDQ employs on average 24% fewer elements as compared to ViChaR and EDVC [13, 55]. An interesting point can be concluded from the results presented in table.

Basically, 49% less critical path delay will allow an RDQ input-port to operate two times faster than the LLD port. The critical path delay of EDVC and RDQ includes the *read-pointer* logic. When the size of input-port buffer increases, the multiplexing stages of EDVC fast *read-pointer* grows exponentially and the increase in the critical path delay of EDVC is also more than RDQ input-port. RDQ approach also consumes one clock cycle less than LLD and ViChaR approaches for flit arrival and departure.

Table 5.6. Input-Port Characteristics for DAMQ Approaches

Type of input-port	ASIC design			FPGA design	
	Area *(μm^2)*	Power *(μW)*[a]	Delay *(ns)*	Combinational *elements*	Registers *(bits)*
LLD 4-slot	1883	48	0.83	95	112(64[b])
ViChaR 4-slot	1917	48	1.07	75	132(64[b])
EDVC 4-slot	1515	35	0.45	86	108(64[b])
RDQ 4-slot	1469	34	0.27	84	108 (64[b])
LLD 8-slot	3305	90	1.26	180	204(128[b])
ViChaR 8-slot	6139	151	1.62	306	392 (128[b])
EDVC 8-slot	3132	72	0.69	206	186(128[b])
RDQ 8-slot	2913	65	0.55	188	186 (128[b])
LLD 16-slot	6147	154	1.16	332	388(256[b])
ViChaR 16-slot	24265	558	2.29	548	896 (256[b])
EDVC 16-slot	6551	146	1.06	441	340(256[b])
RDQ 16-slot	5518	126	0.94	380	340(256[b])
LLD 32-slot	11968	289	2.06	689	769 (512[b])
ViChaR 32-slot	109361	2296	2.42	1183	2040 (512[b])
EDVC 32-slot	14040	311	1.52	964	646(512[b])
RDQ 32-slot	11141	260	0.94	793	646 (512[b])

[a] Frequency for power estimation= 400 MHz.
[b] SRAM registers.

An important characteristic of high scaled CMOS technology like 32/28 nm is that the static (leakage) power supersedes the dynamic power at 400 MHz frequency. For example, the average dynamic power of input-ports includes almost 10% of their total power. This characteristic indicate that the power is more or less proportional to the hardware overhead of a design than its functionality. In other words, the more cells consume more static power and the synthesis results given in Table 5.6 also confirm it. The ViChaR architecture is expensive in terms of hardware cost among all the past architectures (i.e. LLD, ViChaR and EDVC). An extra OR gate per VC in the arbiter (mentioned in Section 5.3.5) is ignored in our comparison due to its tiny hardware usage as compared to the overall input-port hardware.

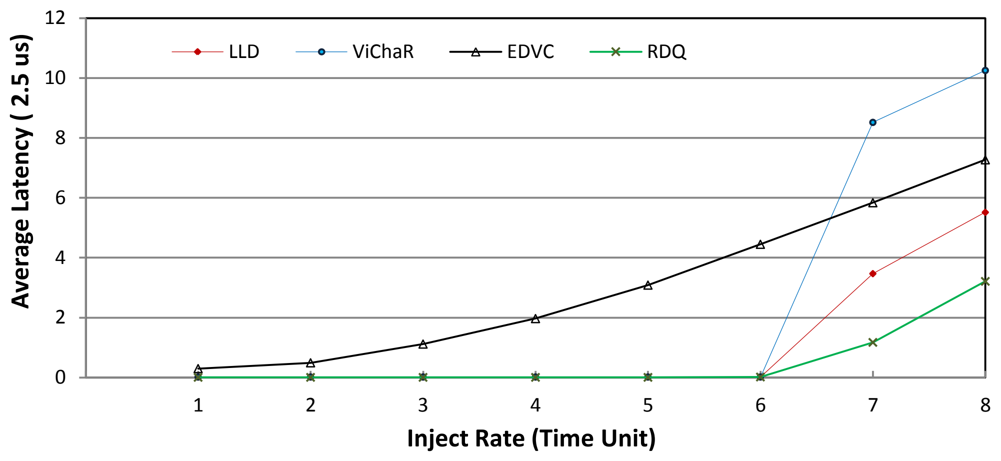
5.5.3 RDQ based NoC Performance

We compare the performance of three DAMQ based VC organization mechanisms (i.e. LLD, ViChaR and EDVC) with the RDQ mechanism. Simulation is performed by employing

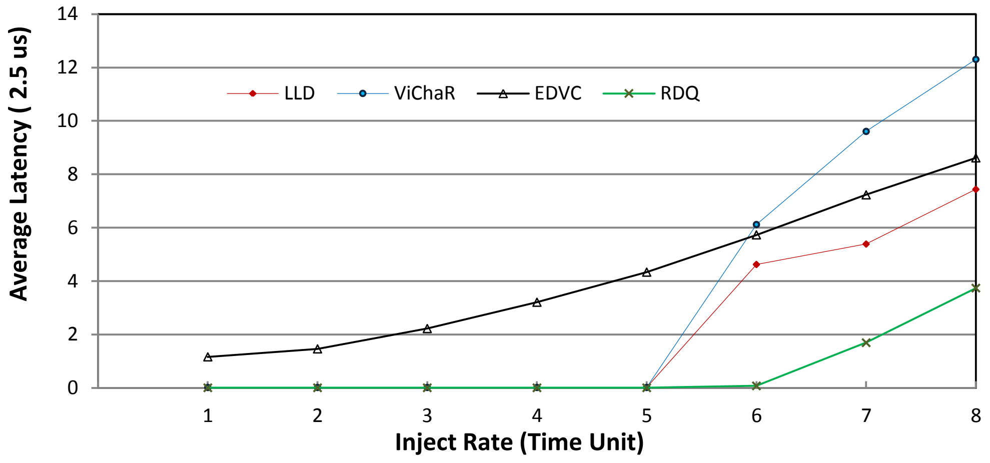
ModelSim to measure various NoC performance metrics. We explore and compare our RDQ approach for 8×8 mesh topologies and for some commonly used traffic patterns such as Uniform-random, Tornado and Complement [16, 55]. Tornado and Complement traffic benchmarks create high contention traffic uniformly in the NoC as described in Section 4.5.2. We have measured the performance metrics of throughput and latency. Throughput is measured by the rate of packets received to the maximum number of packets being injected at a specific time. The average latency is measured by the average time delays associated with the departure and arrival of a specific number (e.g. 2048) of packets in the NoC. The link delay between two routers is negligible as compared to the delay of a router and it is ignored. The communication of packets is based on wormhole switching where the channel width is equal to the flit size (16 bits). A packet consists of 16 flits, and each input-port includes one central 8-slot buffer. There are four VCs per input-port except for ViChaR that has 8 VCs (We have already explained that the number of VCs in ViChaR is equal to the number of input-port buffer slots). The throughput and latency are measured for flit injection rates per time unit (20 ns) and presented in Figures 5.27 and 5.28. For example, flit injection rate 8 means each node (source core) injects 8 flits per 20ns. The flit arrival/departure for RDQ and EDVC routers is one cycle as compared to two cycles for LLD and ViChaR routers.

In spite of higher VC numbers of ViChaR (VC number is equal to the number of slots), the ViChaR has the lower performance as compared to LLD especially at high injection rate. This is because the ViChaR mechanism does not employ any reserved slot for each VC. When the traffic is populated, the probability of monopolizing an input-port by a growing VC is increased. Consequently, the performance of ViChaR decreases as compared to that of LLD.

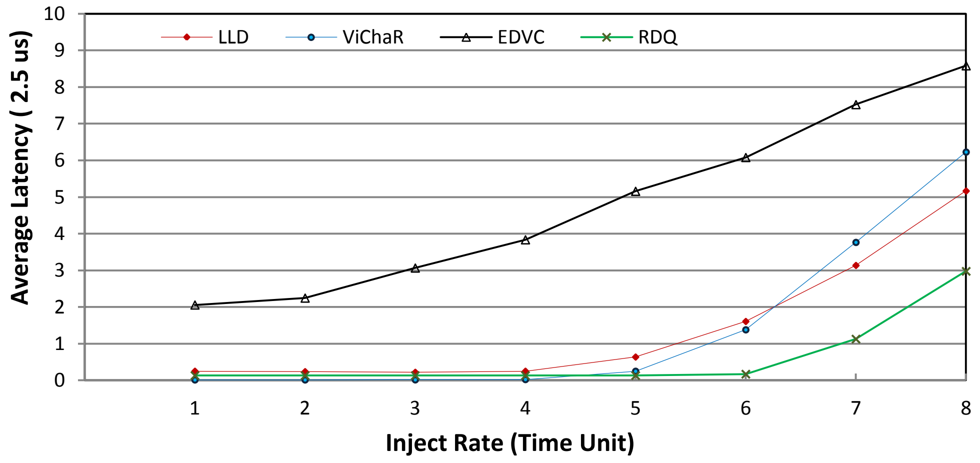
The performance of EDVC is lower than LLD but higher than ViChaR at high injection rates for Complement and Tornado traffics. This is different as compared to the results in Chapter 4 (Figures 4.22 to 4.25). The only difference between two experiments is the *VC-Selectors* utilized by LLD and ViChaR mechanisms. The LLD and ViChaR *VC-Selectors* follow a fixed priority scheme in the experiments of Chapter 4 and a RR priority scheme is used in the experiment here (see Figure 5.21). The performance metrics of LLD mechanism are improved by the RR based VC selection and become better than EDVC. For example, the LLD has 63%, 48% and 70% lower average latency as compared to EDVC at high injection rates for Tornado, Complement and Uniform-random traffic patterns respectively.



(a) Latency (Tornado Traffic)

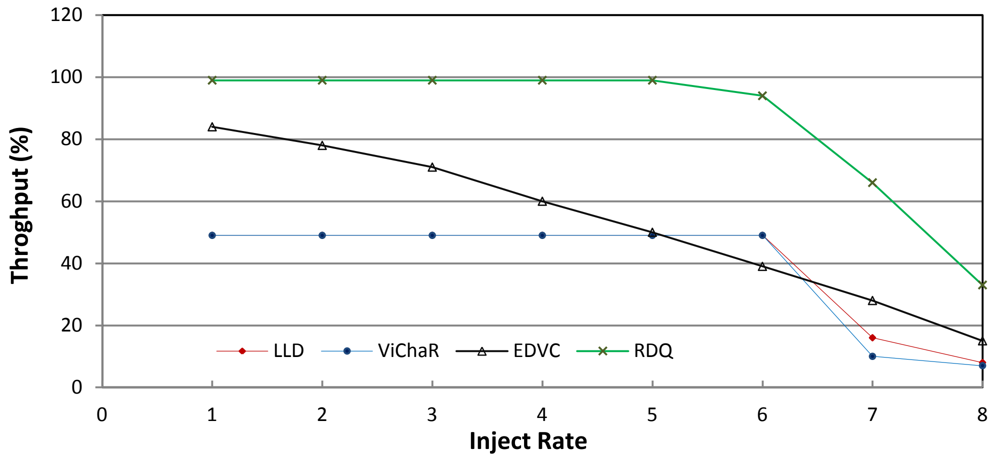


(b) Latency (Complement Traffic)

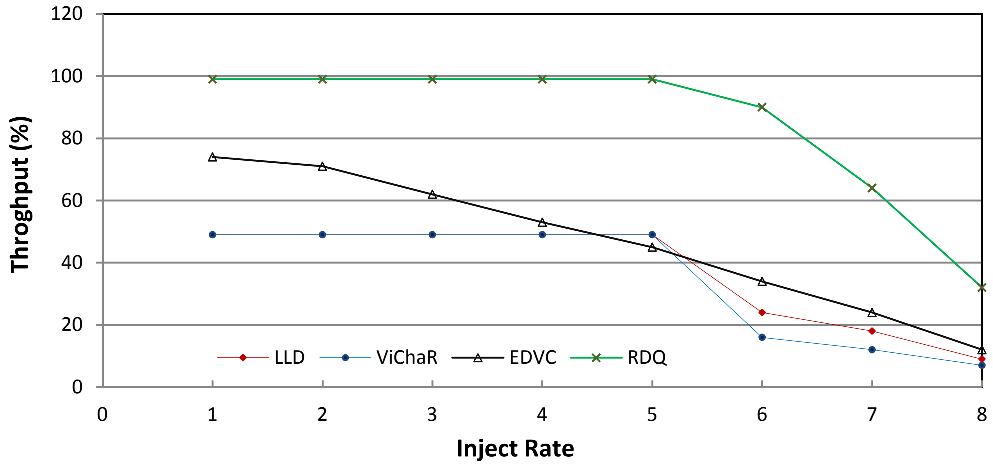


(c) Latency (Uniform-random Traffic)

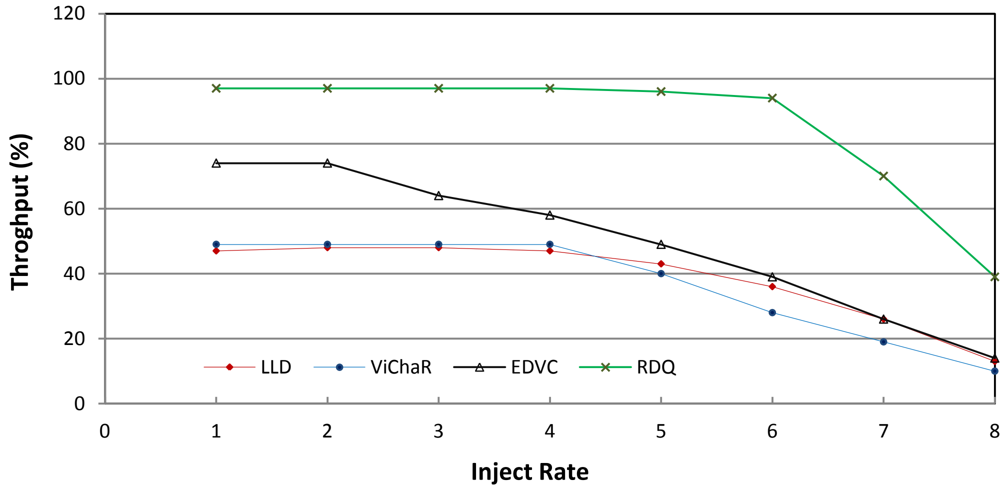
Figure 5.27: Latency for Tornado, Complement and Uniform Random Traffic in 8x8 Mesh Topology.



(a)Througput (Tornado Traffic)



(b) Throughput (Complement Traffic)



(c) Throughput (Uniform-random Traffic)

Figure 5.28: Throughput for Tornado, Complement and Uniform Random Traffic for 8x8 Mesh Topology.

The results of Figures 5.27 and 5.28 indicate the highest performance for RDQ for all the traffic patterns. For example, the average latencies of RDQ are 51%, 68% and 57% less than those of LLD for Tornado, Complement and Uniform-random traffics respectively. The average throughput of RDQ is higher than those of LLD. It is 53%, 56% and 55% higher in Tornado, Complement and Uniform-random traffic patterns respectively. For higher injection rates, more flits are injected, and the NoCs become populated producing higher contention. There is back-pressure associated with the blocked packets in the RDQ communications. The back-pressure at high contention shows some performance improvements [55]. First of all, the probability of a monopoly of an input-port buffer by a growing VC is reduced. Secondly, the free packets receive more buffer free space to pass through the NoC. The EDVC has lower performance as compared to the RDQ. This is due to the fact that the RDQ mechanism has been improved as discussed in Section 5.1.3. The average latencies of RDQ are 82%, 84% and 87% less than those of EDVC for Tornado, Complement and Uniform random traffic patterns respectively.

5.5.4 NoC Evaluation Result

One way to increase the speed of NoC circuits is by using superior technology. However, it can be also achieved by improving the architectural components of NoC system by utilizing simpler and smaller circuits. This may lead to even more impressive improvement in other metrics of NoC systems as we are investigating in the following experimental sections.

We structure eight types of NoCs based on the conventional and novel approaches discussed in this dissertation. The hardware structure and performance of these NoCs are evaluated and compared to illustrate the efficiency of our proposed approaches. As discussed earlier, the NoC architectures follows GALS scheme and the links between routers are assumed to have no effect on the performance and hardware requirements of the NoCs. Therefore, the architectures of router are the main factors in the speeds (clock rate) and the hardware overhead. The architectures of seven NoCs are presented as follows. The first NoC is called RDQ-IRR and includes the RDQ input-port and IRR arbiter in its router architectures. The second NoC is called EDVC-IRR and its structure is based on the EDVC input-port and IRR arbiter architectures. The third, fourth and fifth NoCs utilize LLD input-port and one of the RoR, Matrix or HDRA arbiters in their structures, and we call them LLD-RoR, LLD-Matrix and LLD-HDRA according to the arbiters utilized in routers. The remainder of the NoCs utilize ViChaR input-port and one of the RoR, Matrix or HDRA arbiters, and we call them ViChaR-RoR, ViChaR-Matrix and ViChaR-HDRA based on their utilized arbiters. The crossbar switch module has an identical structure due to the same input-port buffer width (16 bits) and the same NoC topology i.e. 2D mesh.

5.5.5 RDQ-IRR based NoC Hardware Requirements

All the above mentioned eight NoCs are evaluated based on three hardware characteristics including power consumption, chip area, and speed (maximum frequency) that are measured with Synopsys Design Compiler. ASIC technology libraries such as 15nm NanGate are employed to illustrate our evaluation [57]. As discussed earlier, the routers mainly represent the hardware characteristics of the NoC systems in our implementation. Therefore, the hardware parameters related to experimental results are associated with the routers. The setup constrains as well as CMOS technology parameters such as global operating voltage of 0.8V and time period of 1 *ns* (1 GHz) is applied to all the components evaluated. The width of the slot buffer is equal to the flit size of 16-bits.

The hardware characteristics of various modules of the NoC router are evaluated, and the results are presented in Tables 5.7 and 5.8. Table 5.7 results are sorted according to the VC and slot numbers utilized in each input-port. For example, the input-port (*port_LLD_HDRA_4v_4s)* presents the hardware characteristics of a LLD-HDRA input-port utilizing 4 VCs and having four slots in its buffer. The LLD-HDRA input-port is based on LLD mechanism and utilizing HDRA arbiter in its *VC-Selector* (see Figure 5.21). As we mentioned earlier in Section 5.3.4, the EDVC and RDQ input-ports do not utilize separate *VC-Selector*. LLD and ViChaR input-ports that utilize Matrix arbiter consume more chip area but have less critical path delay. The trend among the (router hardware) characteristics shown in Table 5.7 follows the trend given in Tables 5.4, 5.5 and 5.6. The RDQ based input-ports shows the optimum hardware characteristics as compared to the other input-ports with the same number of VCs and buffer slots. An important difference between the LLD and ViChaR input-ports listed in Table 5.6 and Table 5.7 is the (utilization of fixed priority arbiter based) *VC-Selectors* in the input-ports of Table 5.6. The fixed priority arbiter is smaller than the RR arbiter, and we expect that the RDQ input-ports of Table 5.7 will be more effective and efficient.

The main differences among the architectures of arbiters are the RR arbiters utilized in their SA modules, some extra OR gates used in RDQ and EDVC VC-block modules and the number of utilized VCs.

Table 5.7. Input-port Hardware Characteristics

VC number	Slot number	Input-port model	ASIC design 15 nm NanGate Library		
			Area *(μm^2)*	Power[(a)] *(uW)*	Critical path *(ps)*
4	4	port_LLD_RoR_4v_4s	376	172	111
		port_LLD_Matrix_4v_4s	382	173	97
		port_LLD_HDRA_4v_4s	377	183	109
		port_ViChaR_RoR_4s	391	202	135
		port_ViChaR_Matrix_4s	397	206	116
		port_ViChaR_HDRA_4s	392	220	130
		port_EDVC_4v_4s	290	64	70
		port_RDQ_4v_4s	281	59	56
	8	port_LLD_RoR_4v_8s	632	252	123
		port_LLD_Matrix_4v_8s	638	254	112
		port_LLD_HDRA_4v_8s	633	264	121
		port_EDVC_4v_8s	596	96	88
		port_RDQ_4v_8s	566	91	73
	16	port_LLD_RoR_4v_16s	1197	435	147
		port_LLD_Matrix_4v_16s	1202	435	133
		port_LLD_HDRA_4v_16s	1198	453	140
		port_EDVC_4v_16s	1281	175	132
		port_RDQ_4v_16s	1144	159	100
	32	port_LLD_RoR_4v_32s	2409	851	195
		port_LLD_Matrix_4v_32s	2415	847	195
		port_LLD_HDRA_4v_32s	2410	863	195
		port_EDVC_4v_32s	2714	359	168
		port_RDQ_4v_32s	2388	324	117
8	8	port_LLD_RoR_8v_8s	881	392	172
		port_LLD_Matrix_8v_8s	937	437	142
		port_LLD_HDRA_8v_8s	887	421	174
		port_ViChaR_RoR_8s	1194	741	195
		port_ViChaR_Matrix_8s	1237	777	162
		port_ViChaR_HDRA_8s	1198	771	190
		port_EDVC_8v_8s	623	102	89
		port_RDQ_8v_8s	592	95	73
	16	port_LLD_RoR_8v_16s	1424	555	179
		port_LLD_Matrix_8v_16s	1469	595	149
		port_LLD_HDRA_8v_16s	1430	586	173
		port_ViChaR_RoR_16s[(b)]	4578	3016	302
		port_ViChaR_Matrix_16s[(b)]	4814	3217	232
		port_ViChaR_HDRA_16s[(b)]	4589	3054	265
		port_EDVC_8v_16s	1333	185	132
		port_RDQ_8v_16s	1197	167	100
	32	port_LLD_RoR_8v_32s	2667	983	202
		port_LLD_Matrix_8v_32s	2700	1021	202
		port_LLD_HDRA_8v_32s	2673	1024	202
		port_ViChaR_RoR_32s[(b)]	19004	13258	454
		port_ViChaR_Matrix_32s[(b)]	19888	14062	327
		port_ViChaR_HDRA_32s[(b)]	19024	13291	343
		port_EDVC_8v_32s	2820	377	169
		port_RDQ_8v_32s	2488	339	120

(a) Frequency for power estimation= 1GHz; the static power is around 10% of total power.
(b) The number of VCs of ViChaR input-port is equal to the number of slots of input-port.

For example, the arbiter, *IRR_8v* utilizes 8 VC, an IRR arbiter in its SA module, and 5 OR gates in VC-block modules as illustrated in Figure 5.22. Therefore, we expect that the trend among the arbiter characteristics given in Table 5.8 follows the trend among the RR arbiter characteristics of Tables 5.4 depending on the number of VCs. The arbiters, *IRR_WF_4v* and *IRR_WF_8v* consume the optimum power and chip area as compared to other arbiters. An important point is related to the same critical path delays of 4-VC and 8-VC arbiters as shown in Table 5.8 results. This is due to the same RR arbiters (5-input) used in both 4-VC and 8-VC arbiters. In Section 5.3.2, we described that the size of RR *output-arbiters* (used in SA) is equal to the number of input-ports of the router. The crossbar module does not have any critical path delay as it does not utilize any register in its structure (see Figure 1.5).

Table 5.8. Arbiter Hardware Characteristics

VC Number	Router Arbiter Model	ASIC design 15 nm NanGate Library		
		Area *(μm^2)*	Power[(a)] *(uW)*	Critical path *(ps)*
4	Matrix_4v	773	436	36
	RoR_4v	701	377	58
	HDRA_4v	710	432	56
	IRR_4v	712	366	42
	IRR_WF_4v	701	359	40
8	Matrix_8v	2092	877	36
	RoR_8v	2020	817	58
	HDRA_8v	2029	874	56
	IRR_8v	2031	806	42
	IRR_WF_8v	2020	798	40
4 or 8	crossbar	104	35	0

(a) Frequency for power estimation= 1GHz; the static power is around 10% of total power.

One can observe that the critical path delays of the arbiters shown in Table 5.8 are less than those of the corresponding input-ports of Table 5.7 despite their higher area and power characteristics. For example, the critical path of arbiter *Matrix_4v* is almost half of the *port_LLD_Matrix_4v_4s* critical path. This is due to two features of our arbiters. First of all, the SA module is separable, while reduces the SA logic complexity as described in Section 5.3. Secondly, the *input-arbiter*s of SA are accommodated in the input-port as discussed in Section 5.3.2. Therefore, considering the critical path delays of input-ports, arbiters and crossbar switch modules, the critical path delays of input-ports determine the maximum operating frequency,

f_{max} of the routers as listed in Table 5.9. The power and area characteristics of each router given in Table 5.9 is calculated by the summation of 5 input-ports, an arbiter, and a crossbar switch characteristics listed in Tables 5.7 and 5.8. Table 5.9 also lists the advantage rate of RDQ-IRR router as compared to the other routers. One can see that the RDQ-IRR routers have at least 17% less chip area, 45% less power consumption, and 73% faster frequency as compared to the LLD and ViChaR routers for 4-VC and 4-slot implementations. For the 8-VC and 8-slot buffer implementations, they have at least 22% less chip area, 53% less power consumption, and 95% faster frequency as compared to the LLD and ViChaR routers. We ignore the other VC and slot configurations of routers for the hardware implementation results listed in the table.

Table 5.9. Router Characteristics and Advantage Rate

VC Number	Slot Number	NoC Router model	ASIC design 15 nm NanGate Library			RDQ-IRR Advantage Rate		
			Area *(μm^2)*	Power [(a)] *(uW)*	Critical path *(ps)*	Area (saving)	Power (saving)	Frequency (faster)
4	4	LLD-RoR	2684	1272	111	17%	45%	98%
		LLD-Matrix	2787	1336	97	20%	48%	73%
		LLD-HDRA	2699	1382	109	18%	50%	95%
		ViChaR-RoR	2759	1422	135	19%	51%	141%
		ViChaR-Matrix	2862	1501	116	22%	54%	107%
		ViChaR-HDRA	2774	1567	130	20%	56%	132%
		EDVC-IRR	2266	721	70	2%	3%	25%
		RDQ-IRR	2221	696	56	N/A	N/A	N/A
8	8	LLD-RoR	6528	2812	172	22%	53%	136%
		LLD-Matrix	6881	3097	142	26%	58%	95%
		LLD-HDRA	6568	3014	174	22%	56%	138%
		ViChaR-RoR	8093	4557	195	37%	71%	167%
		ViChaR-Matrix	8381	4797	162	39%	73%	122%
		ViChaR-HDRA	8123	4764	190	37%	72%	160%
		EDVC-IRR	5250	1351	89	3%	3%	22%
		RDQ-IRR	5095	1316	73	N/A	N/A	N/A

(a) Frequency for power estimation= 1 GHz; the static power is around 10% of total power.

5.5.6 Performance Evaluation of RDQ-IRR NoC

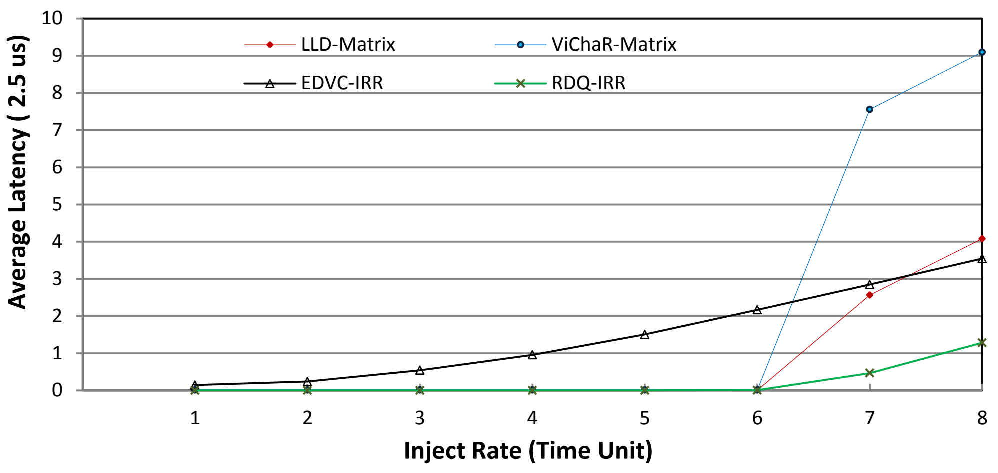
The main metrics of NoC performance are latency and throughput, which are measured for our NoC evaluation by employing the ModelSim platform described in Section 5.5.3. The other setup configurations such as the NoC topology, packet communication, packet structure also follows the experimental setup employed in Section 5.5.3.

We explore and compare various NoCs mentioned earlier such as RDQ-IRR, EDVC-IRR, LLD-RoR, LLD-Matrix, LLD-HDRA, ViChaR-RoR, ViChaR-Matrix and ViChaR-HDRA for

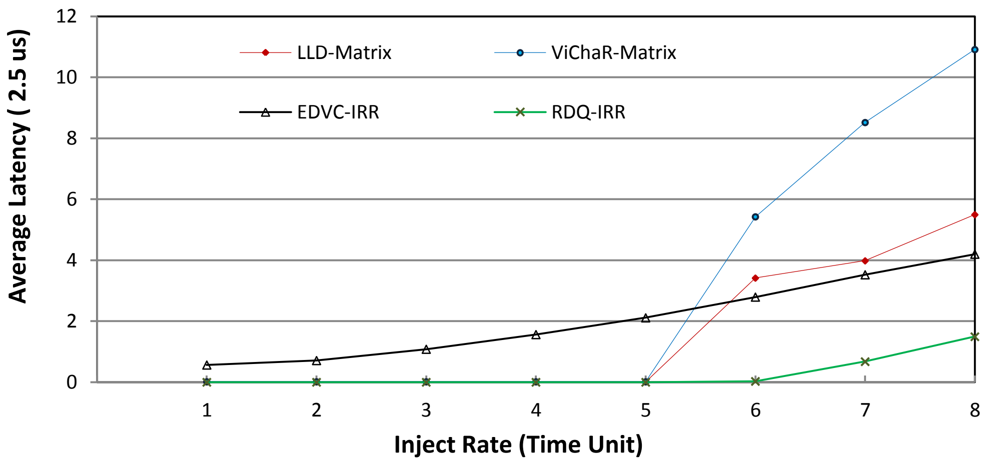
8×8 NoC mesh topologies and for some commonly used traffic patterns such as Uniform-random, Tornado and Complement. In this experiment, the performance metrics of each NoC depends on its functional behaviour of the data flow mechanism and the timing characteristics associated with the router. In other words, the speed of NoC depends on the latency and throughput metrics. The critical path delays associated with the router of each NoC must be considered in the evaluation of NoC performance. Therefore, we test these NoCs under different clock rates according to the critical path delays associated with their routers. In this experiment, the performances parameters of aforementioned NoCs are evaluated and the results are shown in Figures 5.29 and 5.30. The clock rate has linear relation with the performance metrics. For example, if *n* packets passes through an NoC system during *t* times at the *f* clock rate, then at $p \times f$ clock rate, $p \times n$ packets passes through the NoC system during *t* times.

As mentioned earlier, the experimental setup adjustments of the NoCs are the same as described in Section 5.5.3, except that the clock rates of the NoCs are different. Our experiments (not shown in this chapter) indicate that the LLD-RoR, LLD-Matrix and LLD-HDRA behave similar to each other in terms of functionality at the same frequency. This is because the RoR arbiter functionally behaves similar to the Matrix arbiter, and it is very close to the HDRA arbiter that has discussed in Section 5.2.4. Therefore, the LLD-Matrix with faster clock rate supersedes the LLD-RoR and LLD-HDRA NoCs in terms of performance. The same conclusion can be drawn for the ViChaR-MatrixNoC. Therefore, five fast NoCs that include LLD-Matrix, ViChaR-Matrix, EDVC-IRR and RDQ-IRR are selected for evaluation and comparison (see Table 5.9). The LLD-Matrix, ViChaR-Matrix, EDVC-IRR, and RDQ-IRR run at 514, 451, 820 and 1000 MHz clock respectively with 8-VC configuration. These clock rates are corresponding to the critical path delays listed in Table 5.9.

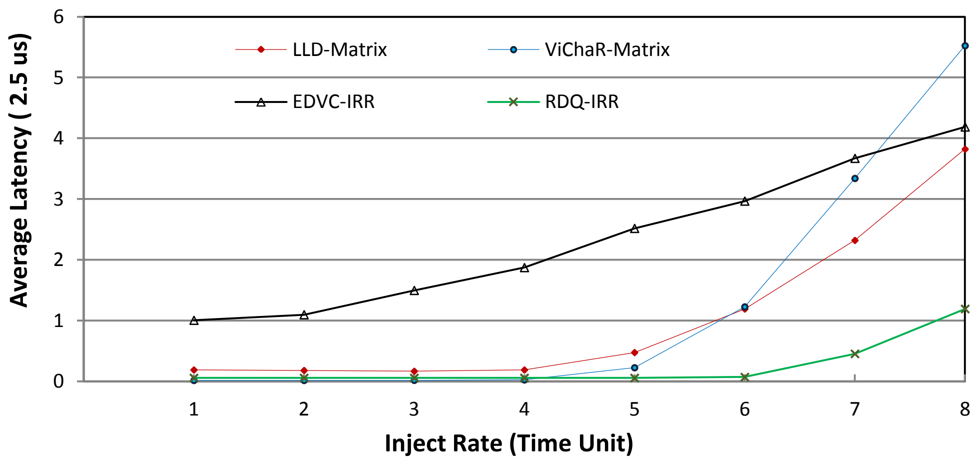
One can expect that the RDQ-IRR results presented in Figures 5.29 and 5.30 shows better performance than those of RDQ illustrated in Figures 5.27 and 5.28. This is due to two reasons. The same advantage discussed for RDQ in Section 5.5.3 is expected for the RDQ-IRR NoC in this section. Moreover, the RDQ-IRR NoC frequency employed for the results presented in this section is higher than those of the other approaches. However, the LLD-Matrix performance in Figures 5.29 and 5.30 become worse than the LLD performance shown in Figures 5.27 and 5.28. This is due to the lower frequency of LLD-Matrix as compared to EDVC-IRR and RDQ-IRR approaches.



(a) Latency (Tornado Traffic)

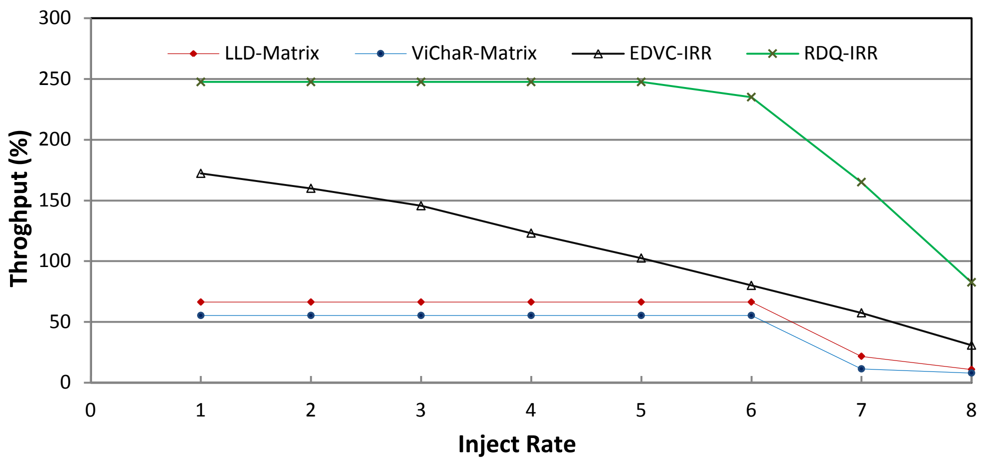


(b) Latency (Complement Traffic)

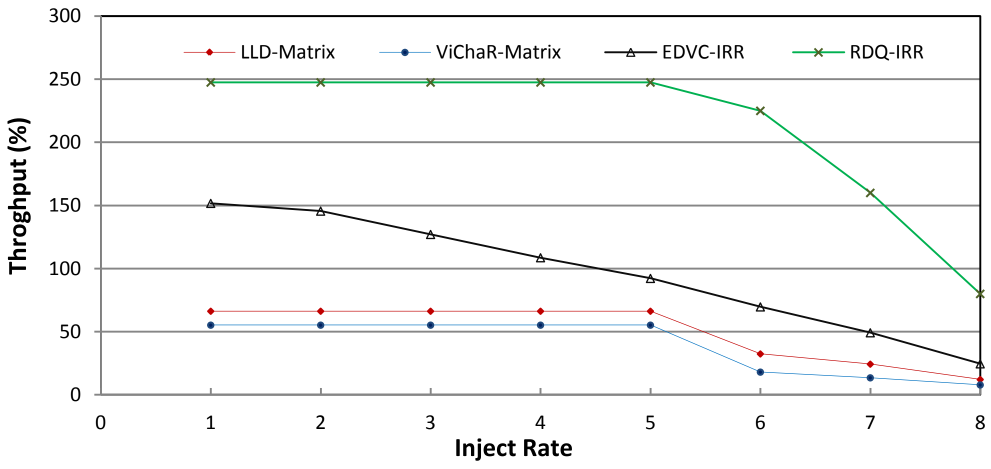


(c) Latency (Uniform-random Traffic)

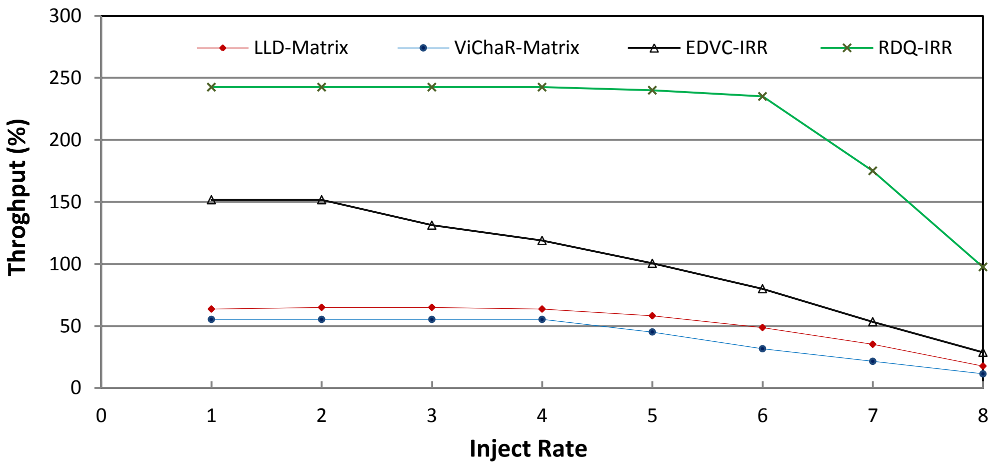
Figure 5.29: latency for Tornado, Complement and Uniform Random Traffic patterns in 8x8 Mesh Topology.



(a)Througput (Tornado Traffic)



(b) Throughput (Complement Traffic)



(c) Throughput (Uniform-random Traffic)

Figure 5.30: Throughput for Tornado, Complement and Uniform Random Traffic patterns in 8x8 Mesh Topology.

The average RDQ-IRR latencies are 85%, 86% and 89% less than those of LLD-Matrix for Tornado, Complement and Uniform-random traffic patterns respectively. The average throughputs of RDQ-IRR are 74%, 76% and 75% higher than those of LLD-Matrix for Tornado, Complement and Uniform-random traffic patterns respectively. The above results also support our conclusions regarding to the results of Sections 5.5.3 and 5.5.6. In other words, the average throughput of RDQ NoC is 55% higher than those of LLD NoC presented in Section 5.5.3, but the average throughput of RDQ-IRR is 75% higher than that of LLD-Matrix NoC presented in Section 5.5.6 for all traffic patterns. Also, the average latency of RDQ is 59% less than that of LLD in Section 5.5.3 when the average latency of RDQ-IRR is 87% less than that of LLD-Matrix in Section 5.5.6 for all traffic patterns.

5.6 RDQ-IRR Router based NoC Features

The main features and advantages of our RDQ-IRR based NoC can be divided into three parts. Firstly, the RDQ input-port provides the following advantage and feature as compared to some previous approaches.

- The RDQ improves our EDVC approach [55] in terms of functionality and lower hardware overhead.
- The RDQ approach improves NoC performance considerably by adding a little hardware.
- The RDQ mechanism employs logic circuits such as rapid *read-pointer, write-pointer* and other blocking circuits instead of tables to manage shared VC slots.
- Our RDQ approach is simpler as compared to table-based mechanisms such as LLD [14, 18] and ViChaR [13]. The main components of the RDQ structure is the read/write pointer circuits, which are scalable that optimize the NoC design and timing performance. The RDQ is faster than table-based DAMQ approaches in terms of arrival/departure time delay. It saves one clock cycle for each flit arrival/departure from input-port.
- There are no configuration constraints in the RDQ approach as compared to other DAMQ mechanisms.
- The RDQ employs a simpler congestion avoidance mechanism.

- The RDQ approach avoids deadlock without reserving in default any buffer space for each VC

Secondly, our IRR arbiter can be utilized for buffering and arbitration in NoC routers to ensure fairness and avoid traffic starvation. It has the following novel features.

- The IRR arbiter has the same functionality as conventional RR arbiters.
- The IRR approach is fast and it requires less hardware as compared to other arbiters.
- The IRR arbiter provides faster arbitration, whereas the arbitration components has dominant role in determining the speed of routers and consequently NoC systems.
- The micro-architecture of IRR arbiter scales logarithmically (log2) with the number of input-ports as compared to a conventional round robin arbiter that scales with the number of input-ports.
- The IRR architecture has the capability to present both strong and weak fairness arbitrations.
- The IRR arbiters consume less power due to fewer numbers of buffers employed.

Finally, we have proposed an efficient and fast DAMQ router architecture called RDQ-IRR whose main features are given below.

- The arbitration among the VCs of RDQ-IRR routers follows a FIFO type operation.
- The arbitration in the switch allocation module of RDQ-IRR routers ensures strong fairness and avoids traffic starvation.
- The RDQ-IRR router arbiter architecture benefits from easy separation or pipeline, and it is much fast that made it suitable for the NoC applications where lower latency is critical.
- The flit arrival/departure in RDQ-IRR routers is faster than that of other table-based DAMQ routers. In addition to saving one clock cycle for each flit arrival/departure from the input-port, the critical path delays of RDQ-IRR router components are lower as compared to the other DAMQ routers.
- In addition to a higher performance RDQ-IRR mechanism, the RDQ-IRR routers are simpler and faster as well as consume lower power consumption and chip area.

5.7 Summary and Concluding Remarks

We have presented our latest NoC router architecture, which is based on new approach called Rapid Dynamic Queue (RDQ). We also presented an efficient and fast arbiter, Index-Based Round Robin (IRR). The effects of a new RDQ input-port and the IRR arbiter have been investigated on the efficiency of NoC systems. The experimental work has been presented, and the results of NoCs utilizing the RDQ and IRR mechanisms have been evaluated and compared with the NoCs utilizing some previous buffering and arbitration techniques. The RDQ-IRR based NoCs have on average 74%, 76% and 75% higher throughputs and 85%, 86% and 89% less latencies than those of LLD based NoCs for Tornado, Complement and Uniform-random traffics respectively.

Chapter 6

Conclusions

NoCs are on-chip communication infrastructures introduced to be utilized in SoC systems. They typically employ packet switching mechanism and VC organization to improve the performance with minimal extra hardware. The VC organizations of NoCs have some drawbacks including complex logic, lower buffer utilization, configuration limitations and HoL blocking. The arbitration is another important component used in NoC router that has some critical drawbacks such as lower speed, weak fairness, traffic starvation, and pipelining problem. In this thesis, we have presented approaches to improve the overall efficiency of NoC rouers and systems.

Some important past research works related to DAMQ based VC organizations and round-robin arbiter architectures are reviewed in Chapter 2. In Chapter 3, we presented an adaptive and efficient VC organization based on Statically Adaptive Multi FIFO (SAMF) [58]. We have explored SAMF architecture and its novel features in detail. Then the experimental work based on SystemC based NoC simulations and Verilog modeling are presented. The SAMF modeling results are compared with some conventional VC (CVC) organizations. A SAMF based 4×4 mesh NoC shows an improvement of 19% for throughput and 23% for latency and it also requires 5% less chip area and 4% less power consumption as compared to CVC NoC for applications with contention traffics.

Efficient Dynamic Virtual Channel (EDVC) organization and its novel features are discussed in Chapter 4 [54, 59]. A 4-slot EDVC input-port consumes on average 10% less registers, 61% less power, and it operates at 10% higher frequency as compared to the LLD (Link-List based DAMQ) input-port for its ASIC design and implementation. EDVC based NoC simulation shows that EDVC mechanism has 48-50% lower latency and 100% higher throughput as compared to LLD approaches for Application-Specific traffic.

A novel and efficient router architecture has been presented and evaluated in Chapter 5. The router utilizes two new components including an RDQ input-port and IRR (Index-Based Round Robin) arbiter. The architecture of RDQ input-port is an improved version of EDVC based input-port [60]. The micro-architecture of RDQ input-port is evaluated and compared with some conventional table-based input-ports (LLD and ViChaR) as well as the EDVC input-port designs. The evaluation results confirm that our RDQ mechanism is efficient in terms of both performance and hardware overhead. An RDQ input-port consumes on average 13% less chip area, 21% less power consumption, and 49% less critical path delay as compared to the LLD input-port implementation. Moreover, the RDQ mechanism improves the throughput by 55%, 59% and 55% as compared to LLD approaches for Tornado, Complement and Uniform-random traffics respectively. A strong fairness index based round robin (IRR) arbiter design is also presented [61]. Our IRR arbiter design provides strong fairness arbitration, which is not guaranteed by some of the earlier designs including HDRA, PRRA and IPRRA. The index based arbitration is simple, fast and requires little hardware overhead. The ASIC level modeling results for 90nm technology show that our IRR arbiter requires 70% less chip area, 74% lower power, and around 43% timing improvement when compared with RoR and Matrix arbiters.

The micro-architectures of routers that utilize our proposed VC organization and arbitration modules i.e. the RDQ input-port and IRR arbiter have been explored. The NoC routers we have designed and implemented are independent in terms of clock rate, and the faster clock rates of routers leads to faster NoCs. We have presented the micro-architectures of our proposed EDVC and RDQ routers in Chapters 4 and 5 and compared them with conventional routers. Among the router modules, the crossbar switch component has an asynchronous architecture. Therefore, it does not affect the speed of router. However, the input-ports and arbiter affects the speed of router due to synchronous buffers. The micro-architectures of arbiter sub-components have been presented, and it is confirmed that the arbitration stages (RC, VA and SA) can be performed in one clock event.

Our state of the art RDQ-IRR router presented in Chapter 5 consumes at least 17% less chip area, 45% less power consumption, and operate at 73% higher frequency as compared to LLD and ViChaR based routers for a 4-VC and 4-slot implementation. Similarly, for 8-VC and 8-slot implementations, the RDQ-IRR routers have at least 22% less chip area, 53% less power consumption, and operates at 95% higher frequency when compared with the LLD and ViChaR based routers. The performance of RDQ-IRR router based NoC has also been modeled and simulated. The average throughput of RDQ-IRR router is 74%, 76% and 75% higher than those of LLD-Matrix (LLD input-port with Matrix arbiter) router for Tornado, Complement and Uniform-random traffic patterns respectively. The average RDQ-IRR latencies are 85%, 86% and 89% lower than those of LLD-Matrix router for Tornado, Complement and Uniform-random traffic.

6.1 Future Work

- Alleviate the latency related to blocking mechanism in RDQ approach. When a blocked VC in RDQ approach becomes freed, the read pointer should point to the first location of buffer maximum two times. This leads to higher latency in NoC. We are going to improve the RDQ mechanism to point one time to the first location of buffer.
- Experiment with the RDQ-IRR NoC approach with different benchmark applications in terms of size and traffic congestion.
- Evaluate the efficiency of RDQ-IRR in terms of hardware and performance by using workloads and traces from existing NoC based SoC architectures.
- Extend the RDQ-IRR router for other NoC topologies including application specific NoCs

References

[1] W.J. Dally and B. Towles. (2004). Buffered Flow Control. In: Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers, pp. 233-256.

[2] W.J. Dally and B. Towles. (2004). Arbitration. In: Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers, pp. 349-362.

[3] L. Benini and G.D. Micheli (2006). Register designs for queuing buffer. In: Networks on Chips: Technology And Tools. San fransisco: Morgan Kaufmann Publishers . pp. 65–66.

[4] Y. Choi and T. M. Pinkston, “Evaluation of queue designs for true fully adaptive routers,” Journal of Parallel and Distributed Computing, vol. 64, no. 5, pp. 606–616, May 2004.

[5] K. Donghyun, K. Kwanho, K. Joo-Young, L. Seung-Jin, and Y. Hoi-Jim, "Solutions for Real Chip Implementation Issues of NoC and their Application to Memory-Centric NoC," in Proc. First International Symposium on Networks-on-Chip, pp. 30-39, Princeton, New Jersey, May 2007.

[6] H.J. Yoo, K. Lee, and J.K. Kim. (2008). Network on Chip based SoC. In: Low-Power NoC for High-Performance SoC Design. Boca Raton: CRC Press. p 142-145.

[7] P. Forstner. (1999). FIFO Architecture, Functions, and Applications. http://www.ti.com/lit/an/scaa042a/scaa042a.pdf Last accessed 2nd Apr 2014.

[8] Y. J. Yoon, N. Concer, M. Petracca, and L. Carloni, "Virtual channels vs. multiple physical networks: A comparative analysis," 47th ACM/IEEE Design Automation Conference (DAC), pp. 162 - 165, Anaheim, CA, June 2010.

[9] W. Danyao, N.E. Jerger, and J.G. Steffan, "DART: A programmable architecture for NoC simulation on FPGAs," Fifth IEEE/ACM International Symposium on Networks on Chip (NoCS 2011), pp. 145 - 152, 2011.

[10] K. Latif, A.M. Rahmani, E. Nigussie, H. Tenhunen, and T. Seceleanu, "A Novel Topology-Independent Router Architecture to Enhance Reliability and Performance of Networks-on-Chip," International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 454 - 462, Vancouver, B.C. Canada, October 2011.

[11] M. Mirza-Aghatabar, S. Koohi, S. Hessabi, and D. Rahmati, "An Adaptive Approach to Manage the Number of Virtual Channels," 22nd International Conference on Advanced Information Networking and Applications, pp. 353 - 358, Gino-wan, Okinawa, Japan, March 2008.

[12] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in Proc. 31st Annual International Symposium on Computer Architecture, pp. 188 - 197, München, Germany, June 2004.

[13] C.A. Nicopoulos, P. Dongkook, K. Jongman, N. Vijaykrishnan, M.S. Yousif, and C.R. Das, "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers," in Proc. 39th IEEE/ACM International Symposium on Microarchitecture, pp. 333-346, Orlando, Florida, Dec. 2006.

[14] M. Evripidou, C. Nicopoulos, V. Soteriou, and J. Kim, "Virtualizing Virtual Channels for Increased Network-on-Chip Robustness and Upgradeability," in Proc. IEEE Symposium on VLSI, pp. 21-26, Amherst, MA, Aug. 2012.

[15] M.A.J. Jamali and A. Khademzadeh, "A new method for improving the performance of network on chip using DAMQ buffer schemes," in Proc. International Conference on Application of Information and Communication Technologies, pp. 1-6, Baku, Azerbaijan, Oct. 2009.

[16] Y. Xu, B. Zhao, Y. Zhang, and J. Yang, "Simple virtual channel allocation for high throughput and high frequency on-chip routers," in Proc. International Symposium on High Performance Computer Architecture, pp. 1-11, Bangalore, India, January 2009.

[17] Y. Tamir and G.L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," IEEE Transactions on Computers, vol. 41, no. 6, pp. 725-737, June 1992.

[18] G.L. Frazier and Y. Tamir, "The design and implementation of a multiqueue buffer for VLSI communication switches," in Proc. IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 466 - 471, Cambridge, Massachusetts, 1989.

[19] T. Yung-Chou and H. Yarsun, "Design and Evaluation of Dynamically-Allocated Multi-queue Buffers with Multiple Packets for NoC Routers," Sixth International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), pp. 1 – 6, Beijing, China, July 2014.

[20] M. Lai, Z. Wang, L. Gao, H. Lu, and K. Dai, "A Dynamically-Allocated Virtual Channel Architecture with Congestion Awareness for On-Chip Routers, " in Proc. 45th annual Design Automation Conference, pp. 630-633, Anaheim CA, USA, June 2008.

[21] M. Lai, L. Gao, W. Shi, and Z. Wang, "Escaping from Blocking: a Dynamic Virtual Channel for Pipelined," in Proc. International Conference Complex, Intelligent and Software Intensive System, pp. 795-800, Barcelona, Spain, March 2008.

[22] D.U. Becker and W.J. Dally, "Allocator Implementations for Network-on-Chip Routers," Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp. 1-12, Portland, OR, 2009.

[23] M. Fattah, A. Manian, A. Rahimi, and S. Mohammadi, "A High Throughput Low Power FIFO Used for GALS NoC Buffers," IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 333 - 338, Lixouri, Kefalonia, Greece, July 2010.

[24] H. Zhang, K. Wang, Y. Dai, and L. Liu, "A Multi-VC Dynamically Shared Buffer with Prefetch for Network on Chip," in Proc. IEEE 7th International Conference on Networking, Architecture and Storage, pp. 320-327, Fujian, China, June 2012.

[25] J. Park and B.W. O"Krafka, S. Vassiliadis and J. Delgado-Frias, "Design and evaluation of a DAMQ multiprocessor network with self-compacting buffers," in Proc. Supercomputing, pp. 713-722, Washington, DC, Nov. 1994.

[26] J.H. Woo, J.H. Sohn, and H.J. Yoo. (2010). Application Platform. In: Mobile 3D Graphics SoC: From Algorithm to Chip. Singapore: John Wiley & Sons (Asia). p 36-37.

[27] H. Wang, L. Peh, and S. Malik, "A technology-aware and energy-oriented topology exploration for on-chip networks," in Proc. Design, Automation and Test in Europe, pp. 1238-1243, Munich, Germany, March 2005.

[28] J. Kathuria, A. Chhabra, G. Kaur, and R. Chadha, "Low power synchronous buffer based Queue for 3D MPSoC," in Proc. World Congress on Information and Communication Technologies, pp. 778-782, Mumbai, India, Dec. 2011.

[29] J. Liu and J. G. Delgado-Frias, "A Shared Self-Compacting Buffer for Network-on-Chip Systems," in Proc. 49th IEEE International Midwest Symposium on Circuits and Systems, pp. 26–30, San Juan, Puerto Rico, August 2006.

[30] J.M. Rabaey, Digital Integrated Circuits, A Design Perspective, Chapter 6: Designing Combinatorial Logic Gates in CMOS, Prentice Hall 1996, 2002.

[31] J. Liu and J. G. Delgado-Frias, "DAMQ Self-Compacting Buffer Schemes for Systems with Network-On-Chip," in Proc. International Conference on Computer Design, pp. 97-103, Las Vegas, June 2005.

[32] C. Nicopoulos, A. Yanamandra, S. Srinivasan, N. Vijaykrishnan, and M. J. Irwin, "Variation-Aware Low-Power Buffer Design," In Proc. Conference Record 41st Asilomar Conference on Signals, Systems and Computers, pp. 1402-1406, Pacific Grove, California, Nov. 2007.

[33] J. G. Delgado-Frias and R. Diaz "A VLSI Self-Compacting Buffer for DAMQ Communication Switches" IEEE Proceedings of the 8th Great Lakes Symposium on VLSI, Lafayette, LA, 1998.

[34] D. Zoni, J. Flich, and W. Fornaciari," CUTBUF: Buffer Management and Router Design for Traffic Mixing in VNET-based NoCs," IEEE Transactions on Parallel and Distributed Systems, vol. PP, no. 99, pp. 1-14 , 2015.

[35] M. Oveis Gharan and G. N. Khan, "Packet-based Adaptive Virtual Channel Configuration for NoC Systems," International Workshop on the Design and Performance of Network on Chip, Procedia Computer Science, vol. 34, pp. 552–558, 2014.

[36] A.T. Tran and B.M. Baas, ”Achieving High-Performance On-Chip Networks With Shared-Buffer Routers,” IEEE Transactions on very Large Scale Integration (VLSI) Systems, vol. 22 , no. 6, pp. 1391 – 1403, 2014.

[37] G. Michelogiannakis and W. J. Dally, “Elastic buffer flow control for on-chip networks,” IEEE Transactions on Computers, vol. 62, no. 2, pp. 295–309, Feb. 2013.

[38] I. Seitanidis, A. Psarras, K. Chrysanthou, C. Nicopoulos, and G. Dimitrakopoulos,“ ElastiStore: Flexible Elastic Buffering for Virtual-Channel-Based Networks on Chip,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. PP, no. 99, pp. 1, Jan. 2015.

[39] G. Jiang, Z. Li, F. Wang, and S. Wei," A Low-Latency and Low-Power Hybrid Scheme for On-Chip Networks," IEEE Transactions on Very Large Scale Integration Systems, vol. 23 , no. 4, pp. 664 - 677, April 2015.

[40] Y. Ben-Itzhak, I. Cidon, A. Kolodny, M. Shabun, and N. Shmuel,” Heterogeneous NoC Router Architecture,” IEEE Transactions on Parallel and Distributed Systems, vol. 26 , no. 9, pp. 2479 – 2492, 2015.

[41] R. Ramanujam, V. Soteriou, B. Lin, and L. Peh, “Design of a high-throughput distributed shared-buffer NoC router,” Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS), pp. 69–78, Grenoble, France, May 2010.

[42] K.A. Helal, S. Attia, T. Ismail, H. Mostafa, "Priority-select arbiter: An efficient round-robin arbiter," IEEE 13th International Conference on New Circuits and Systems (NEWCAS), pp. 1-4, Grenoble France, June 2015.

[43] Z. Fu and X. Ling, "The design and implementation of arbiters for Network-on-chips," 2nd International Conference on Industrial and Information Systems, pp. 292-295, Dalian, 2010.

[44] S. Q. Zheng and M. Yang, “Algorithm-Hardware Codesign of Fast Parallel Round-Robin Arbiters”, IEEE Transactions on Parallel and Distributed Systems, vol. 18, issue 1, pp. 84-95, Jan., 2007.

[45] Y. Lee, J. M. Jou, and Y. Chen, "A High-Speed and Decentralized Arbiter Design for NoC," IEEE/ACS International Conference on Computer Systems and Applications, pp. 350-353, Rabat, 2009.

[46] F. Guderian, E. Fischer, M. Winter, and G. Fettweis, "Fair rate packet arbitration in Network-on-Chip", 2011 IEEE International SYSTEM-ON-CHIP Conference (SOCC), Taipei , pp. 278 – 283, 2011

[47] J. Hyunjun, A. Baik Song, N. Kulkarni, Y. Ki Hwan, and K. Eun Jung, "A Hybrid Buffer Design with STT-MRAM for On-Chip Interconnects," IEEE/ACM International Symposium on Networks on Chip (NoCS), PP. 193 - 200, Copenhagen, Denmark, May 2012.

[48] M.A. Khan and A.Q. Ansari, "n-Bit multiple read and write FIFO memory model for network-on-chip," 2011 World Congress on Information and Communication Technologies (WICT), pp.1322 - 1327, 2011.

[49] W. J. Dally and B. Towles. (2004). Router Datapath Component. In: Principles and Practices of Interconnection Networks, CA: Morgan Kaufmann Publishers, pp. 325-348.

[50] M. Oveis-Gharan and G. N. Khan, “A Novel Virtual Channel Implementation Technique for Multi-core On-chip Communication,” in Proc. IEEE 24th International Symposium on Computer Architecture and High Performance Computing (WAMCA 12), pp. 36-41, Columbia Univ. NY, Oct. 2012.

[51] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, “An 80-tile sub-100-W tera FLOPS processor in 65-nm CMOS,” IEEE Journal of Solid-State Circuits, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[52] V. Dumitriu and G.N. Khan, "Throughput-Oriented NoC Topology Generation and Analysis for Performance SoCs," IEEE Transactions on VLSI Systems, vol. 17 , no. 10, pp. 1433-1446, Piscataway NJ, USA, October 2009.

[53] M. Oveis-Gharan and G. N. Khan, "Flexible Simulation and Modeling for 2D Topology NoC System Design," in Proc. IEEE Symposium Computers, Software and Applications, pp. 180-185, Niagara Falls, Canada, May 2011.

[54] M. Oveis-Gharan and G.N. Khan, " Efficient Dynamic Virtual Channel Organization and Architecture for NoC Systems," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 24 , Issue 2, pp. 465-478, March 2015. http://dx.doi.org/10.1109/TVLSI.2015.2405933

[55] N. Alfaraj, J. Zhang, Y. Xu, and H.J. Chao, "HOPE: Hotspot Congestion Control for Clos Network On Chip," in Proc. IEEE/ACM International Symposium on Networks on Chip, pp. 17-24, Pittsburgh, PA, May 2011.

[56] L. Shaoteng, A. Jantsch, and L. Zhonghai, "A Fair and Maximal Allocator for Single-Cycle On-Chip Homogeneous Resource Allocation," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 22 , Issue 10 , pp. 2229 - 2233, October 2013.

[57] NANGATE. 2014. Nangate Releases 15nm Open Source Digital Cell Library. [ONLINE] Available at: http://www.nangate.com. [Accessed 15 December 15].

[58] M. Oveis-Gharan and G. N. Khan, "Statically Adaptive Multi FIFO Buffer Architecture for Network on Chip," Microprocessors and Microsystems, Volume 39, Issue 1, pp. 11–26, February 2015.

[59] M. Oveis-Gharan and G. N. Khan, "Efficient Virtual Channel Organization and Congestion Avoidance in Multicore NoC Systems," in Proc. International Symposium on Computer Architecture and High Performance Computing Workshop (SBAC-PADW), Paris, pp. 30-35, Oct. 2014.

[60] M. Oveis-Gharan and G.N. Khan, "Dynamic VC Organization for Efficient NoC Communication," in Proc. IEEE 9th International Symposium Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Turin, pp. 151-158, Sept. 2015, http://dx.doi.org/10.1109/MCSoC.2015.12.

[61] M. Oveis-Gharan and G.N. Khan, "Index-Based Round-Robin Arbiter for NoC Routers," in Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Montpellier, pp. 62-67, July 2015, http://dx.doi.org/10.1109/ISVLSI.2015.27.

Glossary

2D	Two-dimensional
Altera	An American manufacturer of Programmable Logic Devices
Arbiter	A component in electronic circuitry that allocates shared resources
ASIC	Application-Specific Integrated Circuit
AV	Audio-Video Benchmark
Crossbar Switch	A switch connecting multiple inputs to multiple outputs in a matrix manner
CVC	Conventional virtual channel method
Complement	Traffic benchmarks create high contention traffic uniformly in the NoC
CQ	Circular queue
DAMQ	Dynamically Allocated Multi-Queue
EDA	Electronic Design Automation software
EDVC	Efficient Dynamic Virtual Channel (our second presented approach)
FAANOS	A Flexible And Accurate NoC Simulator coded in SystemC in our Lab
FCFS	First Come First Serve
FIFO	First-in First-out
f_{max}	Maximum clock frequency where a system can be clocked
FPGA	Field programmable gate array
GALS	Globally Asynchronous Locally Asynchronous
HoL	Head of Line blocking problem
HDRA	High speed and Decentralized Round robin Arbiter presented in [45]
Hotspot	One destination is chosen for all the source cores during a time period
IP	(Intellectual Property core), a reusable design unit owned by one party
IPRRA	Improved Parallel Round Robin Arbiter presented in [44]
IRR	Index-Based Round Robin
IRR_WF	Weak Fairness version of IRR
Link-List	A linear data structure where each element refers to the next element.
LLD	Linked-List based DAMQ
Matrix	A Round Robin arbiter presented by Dally and Towles [2]
MPEG	Moving Picture Experts Group (kind of video format) (MPEG4 decoder)

MPSoC	Multi-Processor Systems-on-Chip
MUX	MultiPlexer Component
NoC	Network on Chip
PRRA	Parallel Round Robin Arbiter presented in [44]
Queue	A type of data structure where stores data in a first come first serve manner
RC	Routing Computation (arbiter sub-component)
RDQ	Rapid Dynamic Queue
RR	Round Robin
RoR	A Round Robin arbiter presented by Dally and Towles [2]
RTL	Register Transfer Level
SMF	Statically Multi FIFO
SMAF	Statically Adaptive Multi FIFO
SoC	System on Chip
SA	Switch Allocation (arbiter sub-component)
ST	Switch Traversal (arbiter sub-component)
Synopsys DC.	A logic-synthesis tool presented by Synopsys Inc.
Table-based	Using a table-based approach to determine shared resource
Tornado	Traffic benchmark that creates high contention traffic uniformly in the NoC
Uniform-Random	Traffic benchmark that creates a random traffic uniformly in the NoC
ViChaR	A table based dynamic multi-queue architecture presented in [13]
VA	Virtual channel Allocation (arbiter sub-component)
VC	Virtual Channel
Wormhole	Wormhole routing is a system of simple flow control in NoC
XY	Routing algorithm: first route horizontally then route vertically
+ve	Positive
-ve	Negative