International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 4, No.1 (Jan-2015)

http://dx.doi.org/10.12785/ijcds/040102

Packet-based Adaptive Virtual Channel Configuration for NoC Systems

Masoud Oveis-Gharan[1] and Gul N. Khan[1]
[1]*Electrical and Computer Engineering, Ryerson University, 350 Victoria St, Toronto, ON M5B 2K3 Canada*

Received 14 May 2014, Revised 15 Jul. 2014, Accepted 16 Aug. 2014, Published 1 Jan. 2015

Abstract: The escalating numbers of on-chip processing cores necessitate the introduction of a high performance and scalable communication backbone. In respond to this, Network on Chip (NoC) systems are introduced to play an important role in determining the performance and power of the entire chip. Specifically, Packet-based NoC is known as the most viable communication solution for the multi-core SoC of the future. In NoC design, the buffering organization directs the control of data flow as well as facilitates the use of Virtual Channels (VC). In terms of buffering, the VC mechanism is categorized into static and dynamic models. In dynamic VC mechanism, VCs employ a variable number of buffer slots according to the on-chip traffic. This feature of dynamic VC mechanism encourages us to introduce the Packet Based Virtual Channel (PBVC) approach. The idea is that a VC is reserved when a packet comes in a router, and released when the packet leaves the router. This prevents a VC to hold more than one packet that subsequently removes the Head-of-Line (HOL) blocking in NoCs. Our proposed technique is more suitable for dynamically allocated multi-queue (DAMQ) schemes. In these schemes, an input or output port comprises a centralized buffer whose slots are dynamically allocated to VCs in real-time and according to the traffic conditions. We introduce the architectural and structural details of our DAMQ buffer organization as well as the hardware that our approach imposes. The simulation results support the theoretical concepts of our proposed technique. The results of the hardware requirements for the proposed model are compared with the conventional models. The experimental results show that PBVC can improve the network latency by 40% and the network throughput by 23% on average as compared to the conventional designs for specific high HOL traffic.

Keywords: Congestion Avoidance, Head-of-Line Blocking, Multi-core NoC Systems, Network-on-Chip, Virtual Channel

1. Introduction

In the NoC domain, wormhole routing is mainly employed for communication among multiple cores. In wormhole routing, packets are divided into small and equal sized flits (flow control digit or flow control unit). The flits of a packet are stored in the channel buffers that are first in first out type. The buffer size of a channel can be less than a packet but equal or more than a flit. When the header flit of a packet passes through a routing path, the route path is reserved by the packet. It means no other packets can utilize that route [1]. This kind of routing cannot avoid traffic congestion when a packet is blocked. The blocking of one packet leads to the blocking of other packets for a channel. This blocking is called HOL (Head of Line) blocking. The HOL blocking reduces the performance in terms of latency, throughput and lower buffer utilization. HOL problem can be alleviated by using Virtual Channels (VCs) [2]. The VC mechanism enables the multiplexing and buffering of several packets for a communication channel [3]. However, VC approach does not remove HOL problem completely [4, 5, 6, 7, 8, 9, 10, 11]. We introduce a method that entirely removes the effect of HOL blocking in a VC based wormhole routing NoC.

A. *FIFO Buffers*

Queue is the main component of a router micro-architecture and it temporarily stores packets in the form of first come first serve (FCFS) order until network resources become available. Two terms, “queue” and “FIFO” sometimes have the same meaning when the concept of first-in-first-out is considered. However, in terms of architecture, FIFO is mostly referred to serial or parallel FIFOs, and the queue is referred FCFS based buffer that also comprises FIFO buffers too. There are two types of FIFO designs and architectural schemes: serial and parallel [2, 12, 13, 14, 15, 16, 17, 18, 19]. The serial

E-mail: Masoud.oveisgharan@ryerson.ca, gnkhan@ryerson.ca

FIFO (such as shift register) that works by fall-through principle is the first generation FIFO. However, the architecture of FIFO is constantly being improved. Currently, most of the FIFOs are parallel, which provides a considerable increase in the number of stored words, along with faster speed [15].

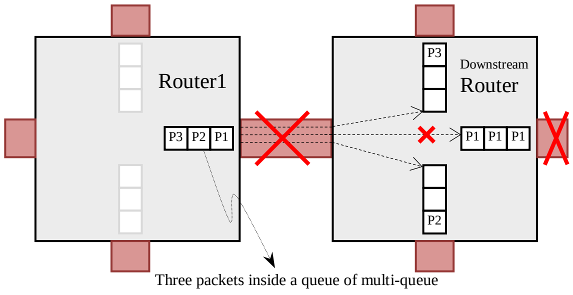


Figure 1: Head of line blocking in DAMQ output port.

B. *Dynamically Allocated Multi-Queue Buffers*

In industrial and academic research, many queue architectures have been proposed and the first-in-first-out queue (FIFO), circular queue (CQ), dynamically allocated multi-queue (DAMQ) and variants of them are well-known queue designs [5, 6, 9, 10, 11, 12]. DAMQ that is a unified and dynamically-allocated buffer structure was originally presented by Frazier and Tamir [5]. It is a single storage array that maintains multiple FIFO queues. DAMQ can be a general solution to the HOL blocking problem discussed earlier [12]. In this technique, packets are stored into the queues of a multi-queue of the output port for routing. Therefore, in the case of blockage of the output port, the packets destined to that output port become blocked. According to Choi and Pinkston, this type of blocking is not HOL [12], but we argue that it is HOL blocking. In fact, HOL blocking can happen inside a queue of a DAMQ buffer [2, 20]. The packets in a queue of output port have different destinations, and they travel to different output ports in the downstream router. Therefore, if the head of line of these packets is blocked due to blocking of its output port in the next downstream router, the remaining packets will be blocked even though their output ports are open in the next downstream router. Figure 1 illustrates a HOL blocking case where eastward output port of router1 is blocked due to blocking of P1. The packets P2 and P3 are blocked despite the fact that their output ports are open in the downstream router.

The DAMQ mechanism dynamically allocates multiple queues over 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 [12]. These dynamic queues of DAMQ buffers lead to a maximum buffer utilization. The DAMQ organization can be used in the VC organization of NoC. This technique is able to solve the other NoC issues such as contention, deadlock or a fault. Jamli et al [6] has used DAMQ buffer scheme for fault tolerance in the NoC systems.

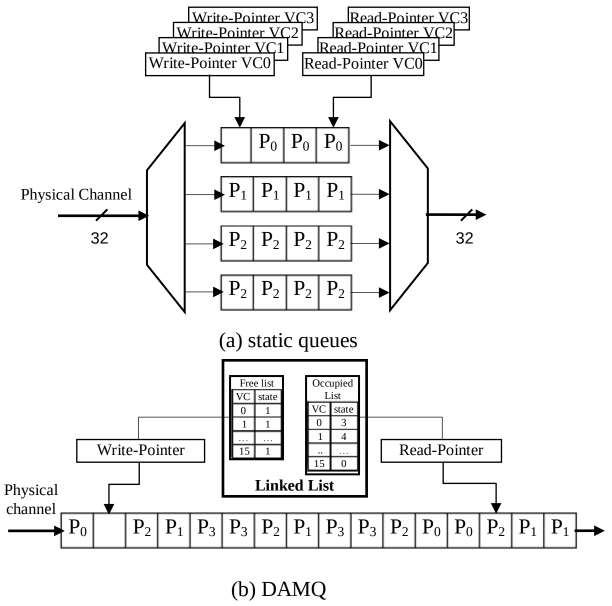


Figure 2: Input port with two queue types.

Before moving ahead, it is better to get familiar with Linked-List based DAMQ mechanism [4]. In Fig. 2, the VC implementation of a physical channel is illustrated through two conventional queue types: static and DAMQ. In a static queue, the buffer slots are statically allocated to incoming packets where in DAMQ queue, the buffer slots are dynamically allocated to incoming packets. The pointers of each queue in Fig. 2a are updated circularly and sequentially for each read and write to the queue. However, DAMQ technique uses linked-list to update the contents of read and write pointers. The linked-list of buffer slots determines the VCs order in the channel buffer as well as the order of slots in each VC [5]. There is a linked-list for each channel that keeps the addresses of occupied buffer slots. Each read pointer is updated according to the occupied list and points to the first slot in the queue. The linked-list is also used to keep track of free slots available for incoming packets. The write pointers are updated according to the free list and point to the slot where incoming flit should be stored. In each read and write to the queues, the linked-list is updated.

Despite the performance merits of DAMQ, it suffers from a few complications and limitations. The first problem is its hardware complexity caused by the linked-list dynamic queue management [12, 20]. The second major problem is that the queue structure is tailored more for deterministic routing algorithms than for fully adaptive routing. In DAMQ, 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 this type of flow control, the routing adaptivity cannot be established [12]. The third problem is that there is no reserved space dedicated for each output port. The packets destined to one specific output port may occupy the whole buffer space, and the new packets destined to this output port have no chance to get into the buffer [6]. The fourth problem is that the DAMQ design uses hardware to implement a linked-list and manage dynamic queue buffer, resulting a larger delay for flit arrival/departure [20]. The remainder of this paper is organized as follows. The previous research works are discussed in Section 2 while our PBVC approach is presented in Section 3. The linked-list based DAMQ buffer is described in Section 4 in terms of its structure and organization. In Section 5, we explain and provides the PBVC experimental results. Finally, the conclusions are drawn in Sections 6.

2. Previous Research Work

Different buffer architectures are proposed to overcome various DAMQ drawbacks. Dynamically allocated multi-queue with recruit registers (DAMQWR) and virtual channel dynamically allocated multi-queue (VCDAMQ) are proposed by Choi and Pinkston [12]. DAMQWR uses DAMQ architecture with recruit registers to implement adaptive routing in NoC. The main function of recruit registers is to allow packets of blocked sub queues employ less congested sub queues. However, in addition to hardware overhead, DAMQWR method also has an additional delay due to recruit register updates and packet recruit operations. VCDAMQ queue organization that resembles DAMQ can efficiently adapt to unbalanced traffic loads among virtual channels by dynamically allocating queue space to virtual channels. In fact, the difference between the VCDAMQ and the traditional DAMQ is that the sub queues of VCDAMQ are associated with router virtual channels while those of the DAMQ are associated with router output ports.

A multi-VC dynamically shared buffer named DAMQ-PF is presented by Zhang et al [11]. Their design has dedicated storage for each output port and a small pre-fetch buffer is used for each VC to store data read from the shared buffer. The same mechanism is also used for the idle address list. In this way, a continuous and concurrent access of the shared buffer is created without any delay. They also proposed a fair credit management method to avoid a situation when a single VC occupies the shared buffer exclusively [11]. Liu et al proposed a DAMQ buffer organization scheme with reserved space for all the virtual channels [7]. The main feature of this scheme is to have a reserved space dedicated for each virtual channel. As shown in Fig. 3, two buffer slots are reserved for each virtual channel before the buffer accepts any incoming flit.

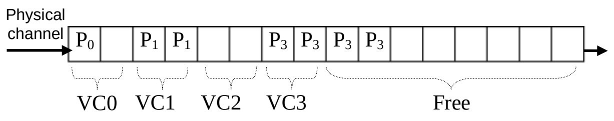


Figure 3: Reserved space for virtual channel [7].

A few researchers have presented the detailed design and implementation of DAMQ buffers. All of these techniques are either expensive in terms of hardware or inefficient due to data dependency (specifically when packet gets bigger). We can observe these problems in the following architectures. A centralized shared buffer architecture called Virtual Channel Regulator (ViChaR) is introduced by Nicopoulos et al [2]. This design avoids using linked list, but still incurs high cost in control logic. ViChaR can support a maximum number of VCs as many as the number of slots in channel buffer. This requires the arbiter in both VC allocation (VA) and switch allocation (SA) stages to be of ***s*** size, where ***s*** is the number of buffer slots. Such size of arbiter can create the latency bottleneck in the critical path of a router that may limit the frequency of NoC [20]. In spite of the advantage of supporting a large number of adaptive VCs, 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 priority for the new packets. Therefore, in case of blocking, a packet can occupy all the slots of a channel buffer and prevents any new packet to pass through the router. If the blocking of that packet continues, the upstream routers will be occupied by the packet and no other packets can pass through the route. This blocking can spread in the entire NoC system and create deadlock. Technically, this problem is due to the design of ViChaR structure where the VC size varies from one to a maximum size channel buffer. 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 Fig. 4. 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. An advantage of ViChaR is that there is no HOL blocking in its communication.

A few interesting features of ViChaR architecture has encouraged some researchers to employ it in their designs. The design of an intelligent buffer that logically reorders the entries in FIFO buffer to minimize overall leakage power consumption is presented by Nicopoulos et al [21]. They employ the ViChaR [2] concept to design their buffer architecture, called IntelliBuffer. In this design, the slots are classified in advance based on their leakage characteristics. Then, the write module always tries to direct incoming flits to the least leaky slots. Moreover, all unused slots are supply-gated using sleep transistors to minimize leakage power consumption. Other mechanisms that employed ViChaR architecture are in the research presented by Xu et al [20]. Their idea is that VCs are assigned based on the designated output port of a packet to reduce the Head-of-Line (HOL) blocking. Unlike ViChaR designs, they only use a small number of VCs to keep the arbitration latency low. In other words, their buffer design is similar to ViChaR except that each VC can store multiple packets, and its VC number is fixed.

Slot availability tracker

Slot	Avail
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 availability tracker

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

(b) 1 denotes the related VC in input port has flit, 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

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

Figure 4: One big table and two trackers used in ViCHar method

The buffer design uses smaller arbiter, however, their VC allocation scheme is between static and dynamic. Two advanced approaches are introduced by Evripidou et al for DAMQ implementation [4]. These two approaches are named Mask-based and Link-List-based mechanisms. The Mask-based approach is cheaper in terms of hardware but slower in performance. In fact, the credit for each VC fallows round robin and synchronized with clock. In other words, a VC waits for a round robin circle and responds at a clock to send its packet. It means that Mask-based communication is synchronous and therefore, the Mask-based approach is not useful for asynchronous communication. One of the important advantages of NoC is to provide asynchronous capability to the system. The Link-List based approach (that mimics DAMQ organization [5]) is expensive in terms of hardware, but it is faster in terms of performance. All of the above DAMQ buffer implementation suffer from HOL blocking except ViChaR that has specific architecture that imposes high latency to the communication and expensive in term of hardware. Therefore, the introduction of a general solution to remove HOL blocking for DAMQ buffers is highly demanded for dynamic VC based NoC systems.

3. Novel Approach In DAMQ Organization

In this section, we present a new mechanism that revises the function of virtual channel in wormhole flow control communication. We name this mechanism as Packet Based Virtual Channel (PBVC). The PBVC mechanism is

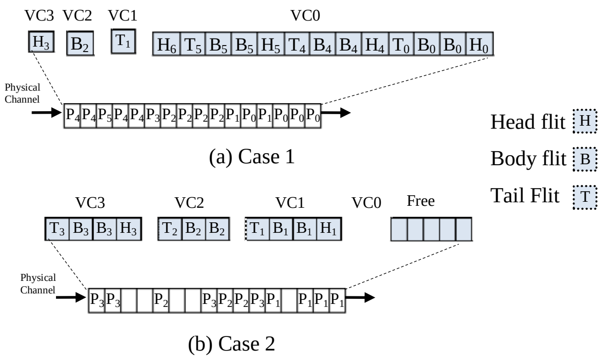


Figure 5: Two different cases in DAMQ buffer (four VCs per channel and four flits per packet) e.g. the four flits of packet P_0 is illustrated by T_0, B_0, B_0 and H_0.

more suitable for dynamically allocated multi-queue (DAMQ) buffers. Therefore, before presenting our PBVC approach, we intend to explain the basis of DAMQ buffer organization and conventional VC mechanism. DAMQ buffers are characterized as their queues (VCs in our proposal) are adapted with traffic condition. In other words, the depth of VC dynamically varies from zero to the size of channel buffer according to the traffic situation. To better describe DAMQ buffers, two different cases are illustrated in Fig. 5. Assume four VCs per channel and four flits per packet. In case 1, the depth of VC0 is thirteen, and in case 2, it is zero. These different depths of VC0 are due to the different traffic requests.

Conventional Wormhole Virtual Channel (CWVC) mechanism is a common approach used in most of the NoC research projects. It is usually used as a basic approach to be compared with the other new approaches [4, 6, 8, 2, 9, 11]. In CWVC mechanism, when the header flit of a packet enters a VC, the VC is reserved by the packet. The reservation of VC is kept till the tail flit arrives. Then the VC can accept a new packet if it has free space. A VC can contain two parts of two packets at a time. The flits are stored in the form of first in first out style in a VC. Assume a VC has two parts of two packets simultaneously. Therefore, blocking of head of line packet blocks another packet in spite of the fact that the route of another packet is open. This is the source of HOL blocking. As shown in Fig. 6, packet#0 is blocked that leads to the blocking of packets#4 and #5. In CWVC mechanism, the HOL blocking leads to a number of problems in NoC such as contention, deadlock, disordering of series of packets and monopoly of whole channel buffer space by a packet. These problems have caused the HoL blocking to become a hot subject in the academia, and it has drawn a lot of attentions of researchers. HOL problem has not solved completely in NoC [4, 5, 6, 7, 8, 2, 9, 10, 11]. Our approach presented in this section provides a complete solution to this problem.

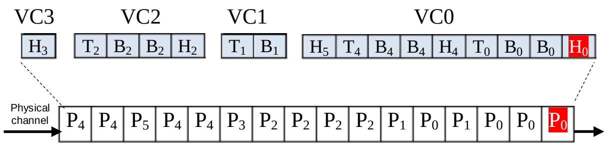


Figure 6: In CWVC, packet #0 is blocked leading to the blocking of packet #4 and #5 (HOL).

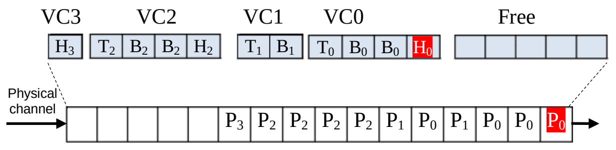


Figure 7: In PBVC, packet #0 is blocked, the remaining buffer space is free to use for the other VCs.

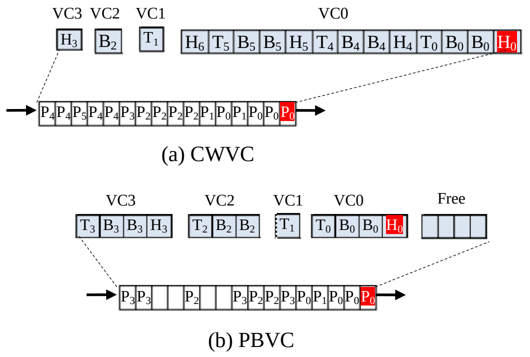


Figure 8: (a) three packets is blocked due to blocking of packet #0 (HOL problem). (b) packet #0 is blocked so occupies only 4 slots, and the free slots can be used for the other VCs.

A. PBVC Approach

In PBVC mechanism, with the advent of header flit of a packet to a VC, the VC is reserved by the packet. Then on the departure of tail flit, the VC becomes free. In fact, the VC does not accept a new packet until there is any flit from the previous packet. This mechanism is different than the conventional (CWVC) mechanism that more than two packets can occupy a VC. Fig. 7 illustrates the PBVC approach from the situation of CWVC buffer shown in Fig. 6. PBVC mechanism has some advantages that can be charaterized as listed below.

- PBVC completely remove the HoL blocking (which arises when some packets occupy a VC and the head of line packet is blocked, the other packets also become blocked) because only one packet can occupy a VC.
- The chance of getting a free VC for the unblocked packets of a VC in our PBVC mechanism is much better than that of CWVC mechanism. For the sake of better understanding, we compare two situations of Figures 6 and 7. Figure 6 represents CWVC situation, where the packet#4 and #5 remain blocked until packet#0 becomes unblocked. In the case of PBVC (hown in Fig. 7), when one of the VC1, VC2 or VC3 becomes free, the packet #4 and #5 can occupy the buffer.
- A little bit of hardware is needed to CWVC structure to create the PBVC architecture. In terms of coding, when the incoming flit of a VC is a tail flit, one “if statement” is required to close the VC. When the departure flit of a VC is a tail flit, one “if statement” is also required to open the VC.
- PBVC approach removes the sequential problem of a series of packets transferred from a source to a destination core. For better explanation, consider a scenario where a series of packets is to be transferred from a source core to a destination, and their route is available. We expect that the packets reach their destinations in a sequential order. In CWVC mechanism, HOL contention can happen in a VC and if a packet of a series is faced with HOL blocking, the remaining packets of the series can go to a free VC and reach the destination before the blocked packet. However, in the PBVC mechanism, as HOL blocking cannot happen, each packet of a series that comes to a channel will move forward and reach the destination in order. In case that a packet is blocked in a channel due to some other reasons, the next packet of series will also be blocked in that channel. Therefore, the series of packets will reach the destination in a sequential order.
- In our PBVC approach, when a packet becomes blocked, its VC gets minimum space in a channel buffer. Consider a situation that a packet is blocked in a VC. In the traditional CWVC scheme, the upstream router continue sending new packets to this VC, and the VC will allocate more buffers and occupies all the free area of DAMQ buffer as illustrated in Fig. 8a. However, in our PBVC approach, the new packets stay in the upstream router until a downstream VC becomes empty. In fact, more free space for the other unblocked VCs is provided in the downstream router as shown in Fig. 8b. Consequently, the performance and buffer utilization of PBVC will be much better than those of CWVC under such conditions.

- There is also an insignificant drawback of our PBVC mechanism. If there is a free VC for a packet, then there is no advantage for it to go to a VC that still has flits. The only negotiable situation happens when the number of requesting packets for a physical channel is more than the number of VCs available for the physical channel. To better clarify the situation, we present the following example. Assume eight packets request for four VCs of a channel. In the CWVC approach, four packets go to four VCs, and four remaining packets will wait in the upstream router. They will stay there until any downstream VC becomes available as illustrated in step #1 of Fig. 9a. When any tail flit of four first packets departs from the upstream router, one of the head flit of waiting packets moves in it. In the downstream router, now two packets occupy a VC (step #2 of Fig. 9a). In PBVC mechanism, the same flow will happen i.e. four packets go for four VCs, and the remaining packets stay in the upstream router waiting for a free VC (step #1 of Fig. 9b). When any tail flit of the first four packets arrives to downstream router, its related VC becomes closed. The VC will be closed when its tail flit departs from the downstream router. We assume a minimum of four tail flits remaining in the downstream router (step #2.1 of Fig. 9b). If one of the tail flits (e.g. T0) departs, its VC (VC0) becomes free (step #2.2 of Fig. 9b). Then a new packet (e.g. H4) can come to the downstream router where three flits (T1, T2 and T3) are still in the channel buffer to be serviced. It indicates that there are always flits in the channel buffer, and the traffic flow of the channel is not interrupted. Consequently, in terms of throughput (rate of transfer), there is no delay of flow in PBVC as compared to CWVC.

Only the buffer utilization of PBVC becomes a little bit lower than that of CWVC, but it will be compensated in most of the situations. In our example (where eight packets at the same time arrive to a 4-VC channel and at the same time depart from the channel), there is one of the large numbers of situations of flow. Mostly packets at different times reach a channel. Thus, due to the adaptivity of DAMQ buffer, the buffer utilization is compensated. As illustrated in Fig. 10, in both CWVC and PBVC buffers VC1 has the flit, T1. In CWVC buffer, VC1 also accepts new packet (P4). However, in PBVC buffer, VC1 doesn’t accept the new packet, but the buffer utilization is the same for both mechanisms. The above explanation clarifies the fact that our PBVC mechanism is not suitable for static buffer based virtual channels. In other words, PBVC mechanism leads to lower buffer utilization or performance when there is no adaptivity for virtual channels. Moreover, in NoCs where the communication involves a large numbers of HOL blockings, PBVC mechanism shows better performance as compared to CWVC.

4. DAMQ BASED VC BUFFER ORGANIZATION

PBVC mechanism is based on the DAMQ approach. In fact, by adding a little hardware to DAMQ structure, we can create the PBVC micro-architecture. Therefore, in order to introduce our PBVC approach, we will first introduce the DAMQ structure. There are different schemes to implement DAMQ method such as Linked-list, Self-Compacting and ViCHaR [2, 4, 8]. Among them, the Link-List based is one of the best schemes to implement DAMQ technique [4]. In this report, we present our DAMQ Link-List based design and call it DLLB. Our DLLB design is coded in Verilog, simulated and implemented in Modelsim for FPGA platforms.

A. DLLB Communication

We employ asynchronous communication between routers, sink and source cores. The advantage of such communication is that the design can be easily implemented in HDL platforms such as FPGA. The asynchronous communication between two sender and receiver routers is achieved through handshaking via credit signals. Figure 11 illustrates an asynchronous communication between two routers in a DLLB scheme, and the following steps describe the communication for a cycle of data transferring.

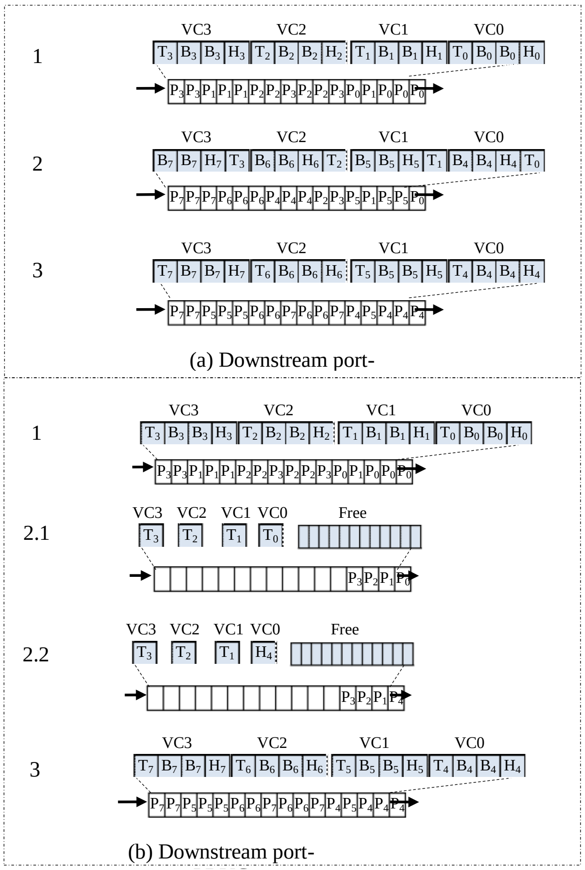


Figure 9: A special situation in CWVC and PBVC buffers where demonstrates that both buffers have the same traffic flow delay.

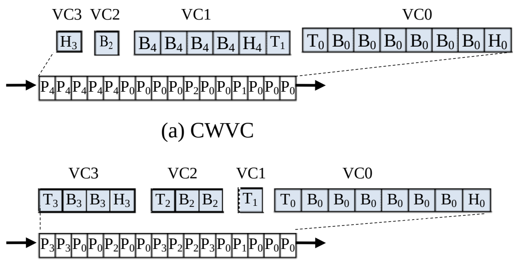


Figure 10: Adaptivity of DAMQ buffers compensates buffer utilization in PBVC, resulting the same buffer utilization for both

#1: In the sender router, a *grant* signal causes the data flit to go out of the sender.

#2: At the negative edge of sender's *grant*, the credit signal for the data is set. The sender's *credit-out* causes two tasks: the data is stored in the receiver's memory, and the receiver's *credit-out* is set.

#3: At the positive edge of sender's clk, the high level of *credit-in* is detected and the sender's, *credit-out* is reset.

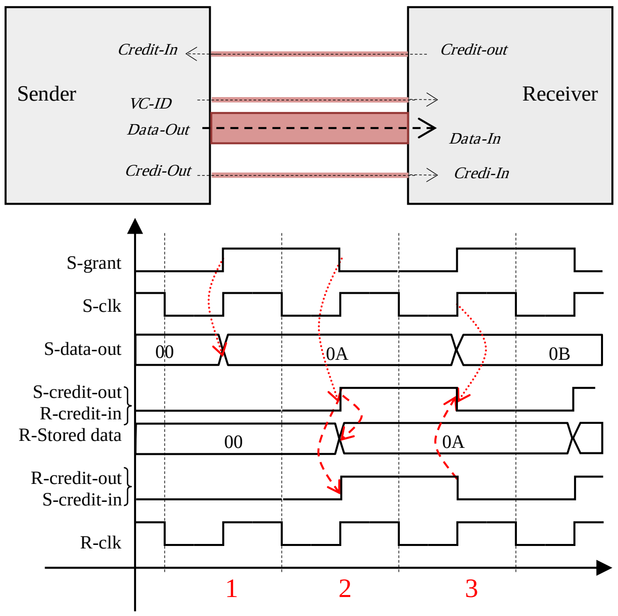


Figure 11: Asynchronous Communication in DLLB approach between two channels of two routers.

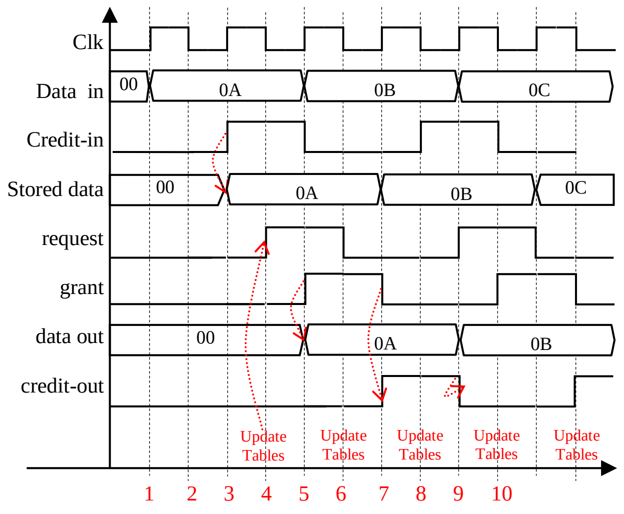


Figure 12: DLLB Asynchronous Communication.

In short, first data and then credit are sent by the sender. In the receiver router, the credit signal causes the data to be stored in the channel buffer and an acknowledgement i.e. *credit out* is sent back to the sender.

The following steps describe the movement of data inside a DLLB router's input port where Figure 12 shows the timing diagram of this movement.

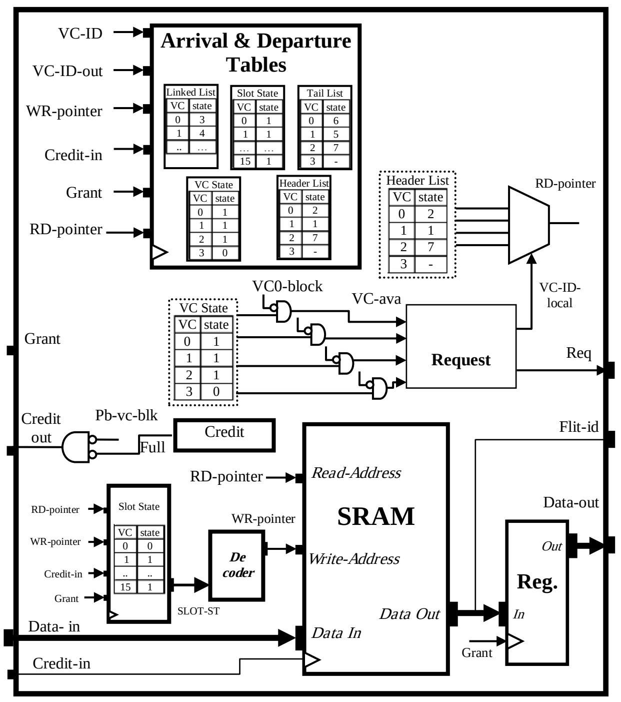


Figure 13: The architecture of DLLB input port.

#1: Data appears at the input-port of the router.

#3: *Credit-in* becomes high that leads the storage of data in the channel buffer.

#4: All lookup tables of the input-port are updated and the request signal is set. The *request* signal causes the *arbiter* to read the data.

#5: After arbitration and at the positive edge of clock, a *grant* signal is issued that lead the data to move out of the input port.

#6: All tables are updated and the *request* signal is reset.

#7: High level of *grant* at the positive edge of clock causes the *credit-out* and the *grant* to be set and reset respectively.

#9: High level of *credit-out* at the positive edge of clock will reset the *credit-out*.

The pipeline communication in Figure 12 shows that a data flit is stored at two clock cycles, and is transferred in two clock cycles for a DLLB input-port. One important feature of the above pipeline is that it can be implemented using any HDL in an FPGA platform. This is because all the tables are updated at negative edge of clock, and the signals are detected and issued at the positive edge of clock.

B. DLLB Router Architecture

The architecture of DLLB input port (i.e. router input port) is illustrated in Figure 13. It contains an SRAM, five tables and some other logic circuits and ports. The SRAM module is the buffer of the channel. The slot size of SRAM is equal to the flit size. Moreover, the data pointed by the *Read-Address* is appeared at the SRAM output. When *credit-in* is active, the data is stored in the SRAM slot pointed by the *Write-Address*. Five tables are used in the DLLB architecture as illustrated in Fig. 14. They are used to implement the Link-List based mechanism. Their functions are briefly described here. The *VC State* table keeps the records of occupied VCs. The *Header List* table has the addresses of channel buffer (SRAM) that point to the header flits of VCs. The *Tail List* table keeps the addresses of SRAM buffers that point to the tail flits of VCs. The *Linked List* table tracks the address of next slot of each buffer slot in the SRAM. In fact, it links the address of flits of each VC in a FIFO manner. The *Slot State* table has the record of occupied slots in the SRAM.

VC State

VC	state
0	1
1	1
2	1
3	0

(a) 0 == VC empty
1== VC occupied

Header List

VC	address
0	2
1	1
2	7
3	-

(b) address of first flit of each VC.

Tail List

VC	address
0	6
1	5
2	7
3	-

(c) address of last stored flit of each VC.

Linked List

Slot	Linked List
0	3
1	4
2	7
3	6
4	5
….	….
15	N

(d) This vector links the addresses of each VC. Sixteen 4-bit registers

Slot State

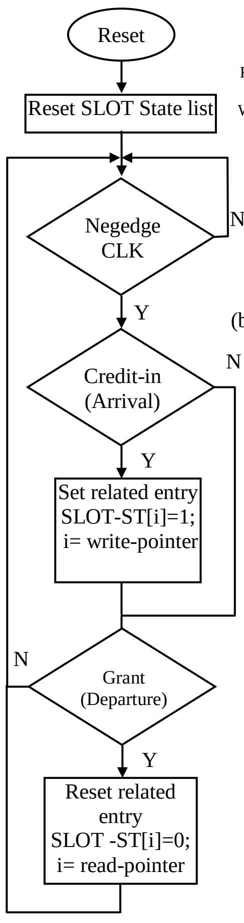
Slot	state
0	0
1	1
2	0
3	1
4	1
….	…
15	1

(e) 0== slot empty.
1== slot occupied.
Sixteen 1-bit registers

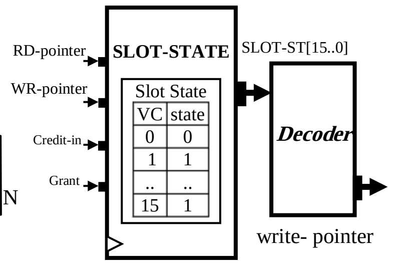
Figure 14: Five tables are used in DLLB Router.

C. Slot State Process

When a flit occupies a slot of SRAM, the corresponding bit of that slot is set in *Slot State* table. When a flit leaves the slot, the corresponding bit of that slot is reset in the *Slot State* table. The content of the *Slot State* table is decoded by the *Decoder* module as illustrated in the pseudo-code of Figure 15d and block-diagram of Figure 15b. The output of *Decoder* is denoted as *write-pointer* and connected to the *Address-Write* port of SRAM. The *Decoder* points to the first unoccupied slot. The flowchart, block-diagram and pseudo-code of *Slot State* and *Decoder* are presented in Figure 15.


Reset
Reset SLOT State list
Negedge CLK
N
Y
Credit-in (Arrival)
N
Y
Set related entry SLOT-ST[i]=1; i= write-pointer
N
Grant (Departure)
Y
Reset related entry SLOT -ST[i]=0; i= read-pointer

(a) Slot State Process Flowchart.



(b) SLOT State and Decoder Block-digram.

```
always @ (negedge clk or posedge rst)
begin : SLOT-ST
  if (rst) begin
              SLOT-ST <=0;
  end else if (credit-in)begin   // data arrival
              SLOT-ST[write- pointer]<=1;
  end if (grant)begin          // data departure
              SLOT-ST[read- pointer]<=0;
   end
end
```

(c) SLOT-ST process pseudo-code.

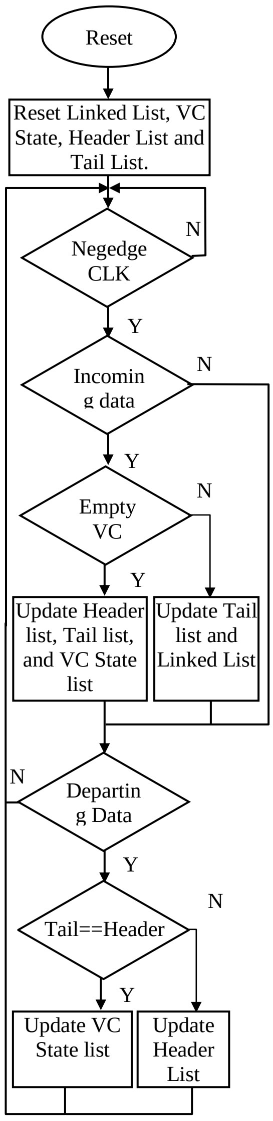
```
  if (!SLOT-ST[0]) begin

               write-pointer =0;
  end else if (!SLOT-ST[1]) begin
               write-pointer =1;
               ……
               …....
  end else (!SLOT-ST[15]) begin
               write-pointer =15;
```

(d) Decoder logic pseudo-code.

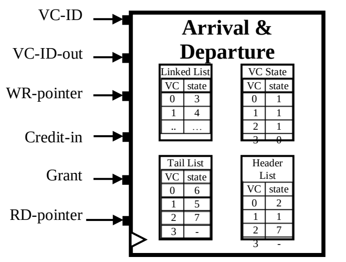
Figure 15: Slot State and Decoder Structure Detail.

D. *Data Arrival and Departure Process*

In this section, we briefly describe the arrival and departure of packet flit in a DLLB input port. We discuss the functions of various tables such as *Header List*, *Tail List*, *VC State* and *Linked List*. Assume a VC (e.g. VC#) is empty and ready to accept a packet flit. On the arrival of a flit to the VC#, the following three tasks occur simultaneously. First of all, the corresponding bit of VC# becomes high in the *VC State* table indicating the VC# is not empty. Then the content of *write-pointer* is stored in the *Tail List* and *Header List* tables. Finally in the *SLOT State* table, the corresponding bit becomes high. The *write-pointer* is updated and points to the next free slot for the incoming flit. Now, assume another incoming flit also tries to occupy the same VC#. As the VC# is not empty, two tasks take place simultaneously. First, the content of *write-pointer* is stored into a location of the *Linked-List* table where the *Tail List* table points. Secondly, the *write-pointer* content is stored in the *Tail List* table. This kind of storage in various tables links the flits of a VC in a FIFO manner. When a flit exits from a VC, e.g. VC#, two conditions may happen. First, if the Header and Tail addresses are the same i.e. this is the last flit, the corresponding bit is reset in the *VC State*. It means that the VC# is empty. Second, if the Header and Tail addresses are not the same, the location of *Linked List* table pointed by the *Header List* table is stored in the *Header List* table. Figure 16 shows the flowchart and pseudo-code of flit arrival and departure module.


Reset
Reset Linked List, VC State, Header List and Tail List.
Negedge CLK
N
Y
Incomin g data
N
Y
Empty VC
N
Y
Update Header list, Tail list, and VC State list
Update Tail list and Linked List
N
Departin g Data
Y
Tail==Header
N
Y
Update VC State list
Update Header List

(a) Process Flowchart.



(b) Block-Diagram

```
always @ (negedge clk or posedge rst)
begin : Arrival Departure
  if (rst) begin
   VC-ST <= 0;
   Header-list  <= 0;
   Tail-list  <= 0;
   Link-list  <= 0;
  end else if (credit-in) begin
    if (!VC-ST[VC-ID])begin
     VC-ST[VC-ID]=1;
     Header-list  [VC-ID]= WR- PTR;
     Tail-list  [VC-ID]= WR- PTR;
    end else begin
      link-list[Tail-list [VC-ID]]= WR- PTR;
     Tail-list  [VC-ID]= WR- PTR;
    end
   end if (grant)begin
    if(Header-list  [VC-ID-out]== Tail-list
[VC-ID-out])
     begin
        VC-ST[VC-ID-out]=0;
        end else begin
              Header-list  [VC-ID]=
                 link-list[Header-list  [VC-ID]];
     end
  end
end
```

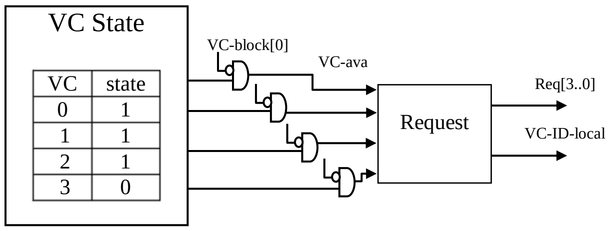
(c) Arrival Departure Pseudo-Code.

Figure 16: Arrival Departure Structure Detail.

E. *VC-Selector Module*

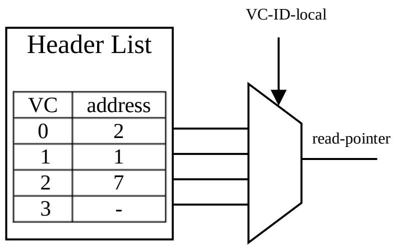
In the router input-port, the *VC-Selector* module selects the VC to be arbitrated. It issues the *request* signal and the *read-pointer* address as illustrated in Figures 17 and 18. The *VC-Selector* module contains a combinational logic circuit that operates on the contents of *VC State* and *Header list*. The combinational logic of the content of *VC*

State table and *VC-block* signals (active when VC is blocked) creates the VC availability signal (*VC-ava*). In fact, *VC-block* signal is reversed and ANDed with its corresponding bit in the *VC State* as shown in Figure 17. For example, if VC0 is blocked, then the *VC-block[0]* signal is high to prevent the selection of VC0. The *Request* module selects a VC depending on the *VC-ava* signals. The *Request* module contains the logic elements based on the following codes.



VC	state
0	1
1	1
2	1
3	0

Figure 17: VC-Selector (Reguest logics).



VC	address
0	2
1	1
2	7
3	-

Figure 18: VC-Selector (Read-pointer logics).

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

The above code illustrates a deterministic scheduling policy that gives first priority to VC0, 2nd priority to 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 and the *read-pointer* is generated as illustrated in Figure 18.

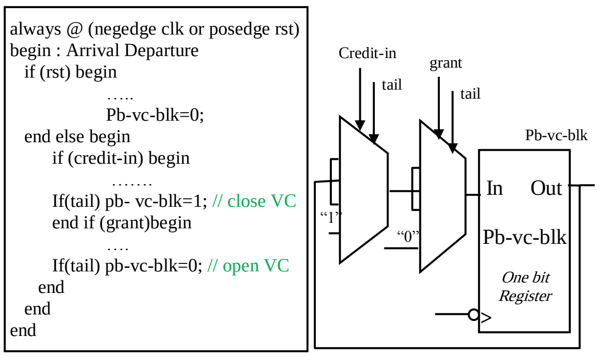
F. VC-block Signal

As shown in Figures 15 and 16, all the tables are updated at the negative edge of router clock. Therefore, the *request* signal and *read-pointer* address is stable for arbitration at the positive edge of router clock and when the *arbiter* module checks the *request* signals. If at least one *request* signal is high, the *arbiter* reads the information of 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 edge of clock. If the requested output is blocked then the *arbiter* issues a block signal (*VC-block*). The *VC-block* causes the *VC-Selector* to select another available VC to be serviced as illustrated in Figure 17.

G. Credit Module

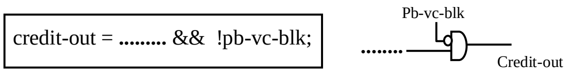
A credit signal is sent to the upstream routers to let them know about the open VC in the downstream router. When the capacity of a VC is full, the credit signal will change to close the VC. The capacity of each VC is dynamic. In our implementation, it varies from one to *M* slots dynamically, where:

M = # of SRAM slots - # of VCs -1.



```
always @ (negedge clk or posedge rst)
begin : Arrival Departure
   if (rst) begin
          .....
              Pb-vc-blk=0;
   end else begin
         if (credit-in) begin
             .......
         If(tail) pb- vc-blk=1; // close VC
         end if (grant)begin
             ....
         If(tail) pb-vc-blk=0; // open VC
      end
   end
end
```

(a) pseudo-code and logics of "if" statements.



```
credit-out = ......... && !pb-vc-blk;
```

(b) Pseudo cods and logics of credit out.

Figure 19: Extra hardware per VC for PBVC Implementation.

For example, assuming sixteen SRAM slots and four VCs, the dynamic capacity of each VC varies from 1 slot to 13 slots. In the DLLB implementation, the credits are regulated only by two conditions as given below.

If {(Current VC == Empty) OR
(# of free Slots < # of free VC)} then credit = ON

In short, a VC is open if it is empty or at least one slot is reserved for each free VC. These two conditions guarantee that at least one slot is dedicated to each VC. Subsequently, the rest of slots are dynamically utilized for all the VCs. This check will remove any starvation and protocol-level deadlocks in the NoC communication.

H. Extra Hardware for PBVC Implementation

As mentioned before, a little bit of extra hardware is required to implement the PBVC approach in DLLB structure. In fact, the coding of two Verilog "if" statements is required to open and close each VC as shown in Figure 19a. The first "if" statement is used to close each VC when the arriving flit is a tail flit. The second "if" statement is used to open the VC when the exit flit is a tail flit. To open and close VC, we use a single bit register, *pb-vc-blk*. It will be set in the case of open and reset otherwise. The output of *pb-vc-blk* is reversed and ANDed with the *credit out* signal of the related VC. Therefore, when *pb-vc-blk* is set, the VC will be closed as shown in Fig. 19b. We also experiment and evaluate the efficiency of PBVC mechanism as compared to CWVC mechanism in DLLB structure and presented in the following section.

5. SIMULATION AND EXPERIMENTAL RESULTS

This section evaluates the efficiency of our PBVC approach as compared to the conventional DAMQ technique. The conventional DAMQ mechanism follows the CWVC approach, and we have used CWVC term for conventional DAMQ. In this experiment, we did experiment for three different traffic patterns including Random Traffic, Fixed Traffic and Special Traffic [22]. Random Traffic is defined where all the destinations are chosen randomly. In the Fixed Traffic, the destination selection is fix and far from the source cores. Special Traffic is a pattern that we have chosen to create a situation with higher Head-of-Line (HOL). By evaluating the results of these three traffic patterns, we will demonstrate the efficiency of our PBVC approach.

A. *Experiment Setup*

We setup our simulator for PBVC and CWVC modes for DAMQ Link-List based (DLLB) architecture. Then we change the number of traffic packets or virtual channels to measure a couple of important performance metrics such as throughput and latency. We did not compare the hardware requirements of these two models as the PBVC model can be created by adding a small hardware to the CWVC model (as discussed earlier in Section 5). The NoC topology selected is a 4×4 mesh, and the communication of packets follows XY routing mechanism as shown in Figure 20.

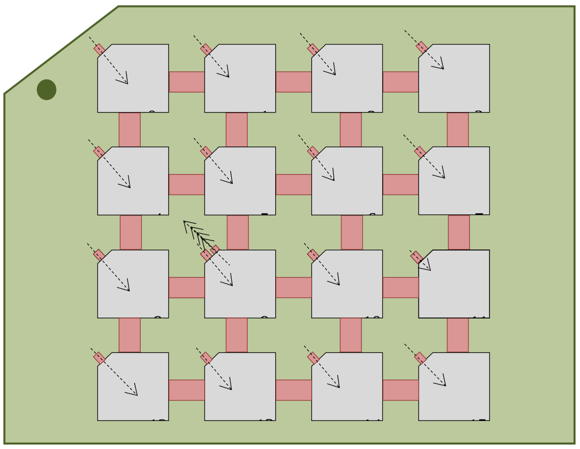


Figure 20: 4×4 Mesh NoC.

The communication of packets is in form of parallel wormhole routing where the channel width is equal to the flit size of 32 bits and each packet is made of 16 flits. The depth of SRAM buffer for each input-port is fixed to 16 slots, where each slot stores a flit. A flit is sent or received by a source core or a router in two clock cycles (Fig. 12). We assume that the time delays of links between routers are negligible as compared to the router delay. Therefore, the time delays of links are ignored in our experiment. The performance of PBVC is compared with the CWVC during these two experiments. In the first experiment, two traffic patterns, Random and Fixed Traffics are applied [23]. In this case, all the sources, destinations and routers are clocked at the same rate (e.g. 1 nsec). In the second experiment, the Special Traffic pattern is applied, where all the source modules send their first packet to a destination (e.g. destination #9 in Fig. 20). The destination #9 is set to be two times slower than the other destinations. After sending the first packet, the rest of the packets of all the sources are sent randomly to all the destinations. We expect that the input-port buffers of router #9 are occupied and will deliver packets slowly at the start. This condition increases the HOL blocking especially when the second packets are being transferred. Our simulator is coded in Verilog and simulation is done by using the ModelSim for an Altera FPGA Platform.

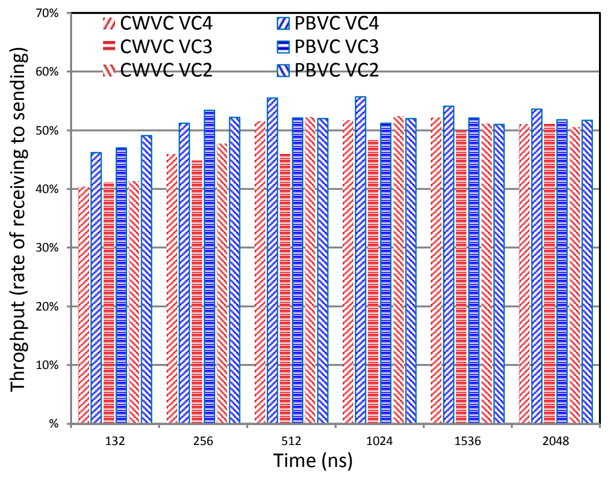


Figure 21: Throughput in Random Traffic.

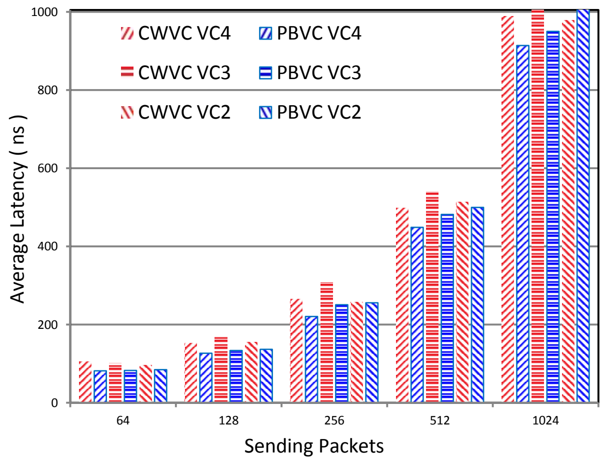


Figure 22: Average Latency in Random Traffic.

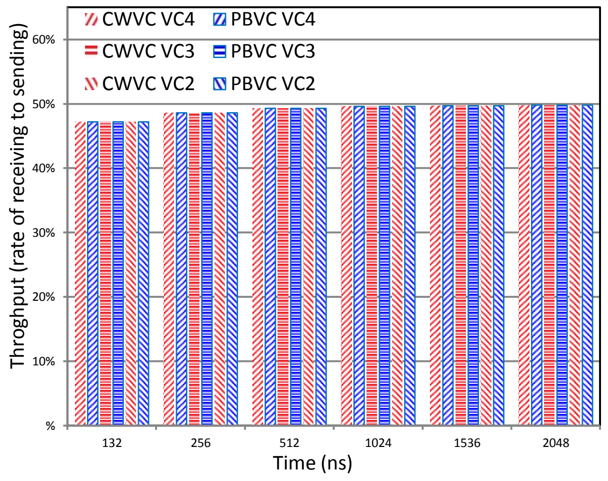


Figure 23: Throughput in Fixed Traffic.

B. Performance Results

In these experiments, the throughput is measured by the rate of receiving packets to the maximum number of packets being sent at a specific time. In other words, at a specific time the NoC that receives more packets is faster in terms of throughput. The latency is measured through the time that a specific number of packets are sent and received by the NoC. Figures 21 and 22 show the throughput and latency results in the case of Random Traffic. In the beginning of simulation (for 132 nsec), the performance of PBVC is much higher than that of CWVC, and as the time passes this advantage diminishes. This is due to the fact that in the beginning of simulation, the traffic is not crowded, and when the HOL blocking occurs in a channel, the incoming packet can move out of the channel. This situation will improve the performance of PBVC mode. On average, the performance of PBVC is better than that of CWVC. For example, in the case of four virtual channels (VC4), the PBVC throughput is2.6% higher and latency is 8.2% lower than those of CWVC.

Figures 23 and 24 provide the throughput and latency results of Fixed Traffic pattern (a source core sends the packets to a farther-away destination). The performance of PBVC is the same as that of CWVC for all the VCs. For this traffic pattern, each source sends packet to a destination, and each destination receives packet from a source. Therefore, there will be very low contention, and PBVC technique will not show any improvement in the performance. In Figures 23 and 24, the results show the fact that when the contention is low, the PBVC approach does not have any drawback as compared to CWVC approach.

In the second part of the experiment, both models are evaluated in a high contention environment. Figures 25 and 26 show the throughput and latency results for the Special Traffic pattern where two conditions are applied. In the first condition, the first packets of all the sources are intended for destination #9, and afterward the packets travel to all the destinations randomly. In the second condition, the destination #9 is two times slower than the other destinations. In this traffic pattern, the PBVC performance improvement is much better than that for the previous two traffic patterns. In fact, by sending 1024 packet, the average latency is 40% less than CWVC, and the average throughput is 23% higher than CWVC for 2048 nsec. This is due to higher HOL blockings occurring in the beginning of simulation. Therefore, in the beginning, the throughput of PBVC is higher than that of CWVC. As the time passes the occurrence of HOL blockings will reduce, and the throughputs of two methods are going to be close to each other. It is obvious that if such traffic pattern is repeated every 164 ns, then the average throughput in PBVC is around 200% higher than the CWVC. Another important point is that as the number of VCs is reduced, the advantages of PBVC will diminish. This is due the fact that when an HOL blocking occurs in PBVC and there are free VCs, the new packet passes through and improves the performance. Free VCs will be mostly available when we have higher number of VCs.

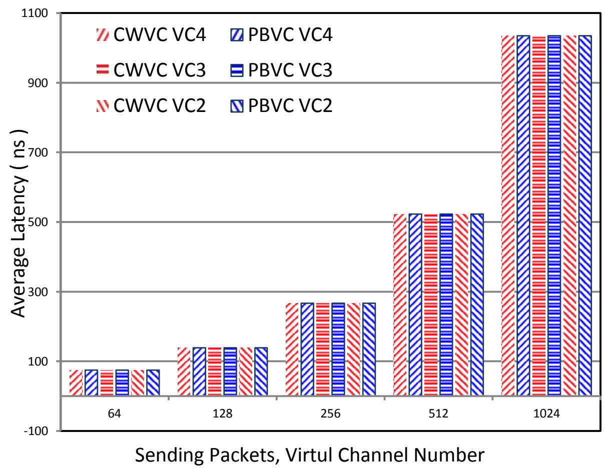


Figure 24: Average Latency in Fixed Traffic.

6. Conclusion

The architecture for a packet-based virtual channel (PBVC) approach is presented in detail. It is argued that PBVC buffer belongs to the family of dynamically allocated multi-queue (DAMQ) buffers and it is able to completely remove HOL blockings in NoC. For this reason, the PBVC and the conventional wormhole VC mechanism (CWVC) are implemented using DAMQ buffers. In the experiments, three traffic patterns i.e. Random Traffic, Fixed Traffic and Special Traffic are applied to PBVC and CWVC based NoCs. Two important NoC metrics i.e. throughput and latency for PBVC approach is compared with those of CWVC. The PBVC performance results are different for three traffic patterns, but the PBVC results are better on average as compared to the CWVC. For Special Traffic pattern, the average latency (40%) and the average throughput (23%) are improved in PBVC as compared to CWVC.

Acknowledgment

This research is partly supported by a grant from NSERC Canada and equipment from CMC. Financial support from the Electrical and Computer Engineering Department at Ryerson University is also acknowledged.

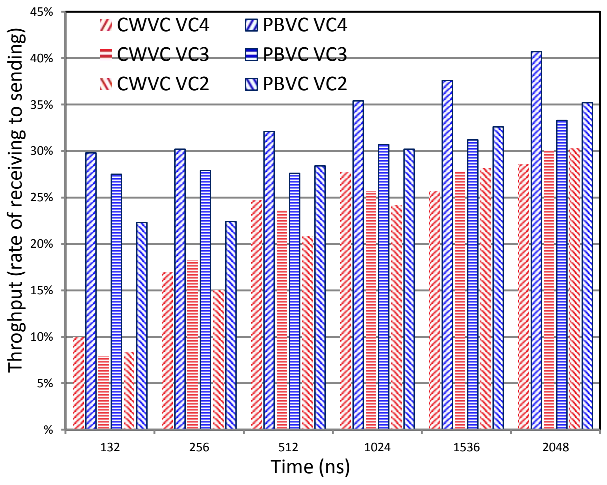


Figure 25: Throughput in Special Traffic.

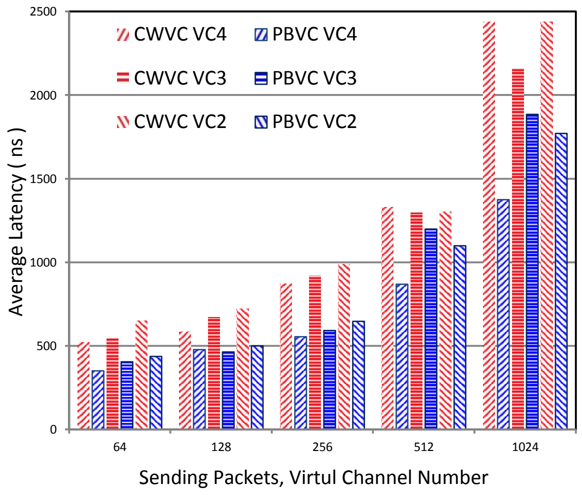


Figure 26: Average Latency in Special Traffic.

References

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

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

[3] W. J. Dally,"Virtual-channel flow control," IEEE Transactions on Parallel and Distributed Systems, pp.194–205, Mar 1992.

[4] M. Evripidou, C. Nicopoulos, V. Soteriou, J. Kim, "Virtualizing Virtual Channels for Increased Network-on-Chip Robustness and Upgradeability," IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 21-26, 2012

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

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

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

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

[9] J. Park, B.W. O"Krafka, S. Vassiliadis and J. Delgado-Frias, "Design and evaluation of a DAMQ multiprocessor network with self-compacting buffers," Proceedings Supercomputing '94, pp. 713 - 722, Nov. 1994.

[10] Y. Tamir and G.L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," IEEE Transactions on Computers, Volume: 41 , Issue: 6 , pp. 725 - 737, June 1992.

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

[12] Y. Choi and T. M. Pinkston, "Evaluation of queue designs for true fully adaptive routers," In Journal of Parallel and Distributed Computing, Volume 64, Issue 5, pp. 606–616, Orlando, FL, May 2004.

[13] L. Benini, G.D. Micheli . (2006). Register designs for queuing buffer. In: Networks on Chips: Technology And Tools. San fransisco: Morgan Kaufmann Publishers . p65 -66.

[14] K. Donghyun, K. Kwanho, K. Joo-Young, L. Seung-Jin, Y. Hoi-Jim, "Solutions for Real Chip Implementation Issues of NoC and Their Application to Memory-Centric NoC," First International Symposium on Networks-on-Chip (NOCS), PP. 30 - 39, Princeton, New Jersey, May 2007.

[15] P. Forstner. (1999). FIFO Architecture, Functions, and Applications. Available: http://www.ti.com/lit/an/scaa042a/scaa042a.pdf. Last accessed 21th Aug 2012.

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

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

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

[19] By H.J. Yoo, K. Lee, 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.

[20] Y. Xu, B. Zhao, Y. zhang, and J. Yang, "Simple virtual channel allocation for high throughput and high frequency on-chip routers," International Symposium on High Performance Computer Architecture (HPCA), pp.1-11, Bangalore, India, January 2010.

[21] C. Nicopoulos, A. Yanamandra, S. Srinivasan, N. Vijaykrishnan and M. J. Irwin, "Variation-Aware Low-Power Buffer Design," Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, pp. 1402 - 1406, Pacific Grove, California, November, 2007.

[22] M. Oveis-Gharan and Gul N. Khan, "A Novel Virtual Channel Implementation Technique for Multi-core On-chip Communication," IEEE 24th Int. Symp Computer Architecture and High Performance Computing (WAMCA 12), Columbia Univ. NY, pp. 36-41 October 24-26, 2012.

[23] M. Oveis-Gharan and Gul N. Khan,"Flexible simulation and modeling for 2D topology NoC system design," IEEE CCECE 2011: Symposium on Computers, Software and Applications, Niagara Falls, Canada, May 2011.

Mr. Masoud Oveis-Gharan received his Bachelors of Engineering in the field of Electrical Engineering (Electronics) from Isfahan University of Technology, Esfahan, Iran in 1991. He completed his Masters of Sciences in the field of embedded system design from Ryerson University, Toronto in 2011. He is currently a PhD student at Ryerson University. His research interests include embedded system design and modeling, computer architectures, logic circuit design and power and performance optimization in Network-on-Chip architectures.

Dr. Gul N. Khan graduated in Electrical Engineering from University of Engineering and Technology, Lahore in 1979. He received his M.Sc. in Computer Engineering from Syracuse University in 1982. After working as research associate at Arizona State Univ. Tempe Arizona, he joined Imperial College London and completed his Ph.D. in 1989. He joined RMIT University, Melbourne in 1993.In 1997, he joined the computer engineering faculty at Nanyang Tech. University, Singapore. He moved to Canada in 2000 and worked as Associate Professor of computer engineering at University of Saskatchewan before joining Ryerson University. Currently, he is a Professor and program director of computer engineering at Ryerson University. His research interests include embedded systems, hardware/software codesign, MPSoC, NoC, fault-tolerant systems, high performance computing, machine vision and multimedia systems.