Chapter 3
ViChaR: A Dynamic Virtual Channel Regulator for NoC Routers [39]

Router buffers are instrumental in the overall operation of the on-chip network. Besides performance, buffers greatly affect the network's overall energy budget. In fact, of the different components comprising the interconnection fabric of SoCs, buffers are the largest leakage power consumers in an NoC router, consuming about 64% of the total router leakage power [50]. Similarly, buffers consume significant dynamic power [22,51] and this consumption increases rapidly as packet flow throughput increases [51]. In fact, it has been observed that storing a packet in a buffer consumes far more energy than transmitting the packet [51]. Furthermore, the area occupied by an on-chip router is dominated by the buffers [14,52,53]. Consequently, buffer design plays a crucial role in architecting high performance and energy efficient on-chip interconnects, and is the focus of this section.

This chapter introduces a novel unified buffer structure – the dynamic Virtual Channel Regulator (ViChaR) – which dynamically allocates buffer resources according to network conditions.

3.1 Importance of Buffer Size and Organization

Decreasing the buffer size arbitrarily to reclaim silicon area and minimize power consumption is not a viable solution, because of the intricate relationship between network performance and buffer resources. Buffer size and management are directly linked to the flow control policy employed by the network; flow control, in turn, affects network performance and resource utilization. Whereas an efficient flow control policy enables a network to reach 80% of its theoretical capacity, a poorly implemented policy would result in a meager 30% [54]. Wormhole flow control [55] was introduced to improve performance through finer-granularity buffer and channel control at the flit level instead of the packet level (a flit is the smallest unit of flow control; one packet is composed of a number of flits). This technique relaxes the constraints on buffer size at each router, allowing for a more efficient use of storage space than store-and-forward and virtual cut-through [56] switching. However, the channel capacity is still poorly utilized; while the buffers are allocated at the flit level, physical paths are still allocated at the packet level. Hence, a blocked packet can impede the progress of other packets waiting in line and may also cause multi-node link blocking (a direct consequence of the fact that the flits of a single packet are distributed across several nodes in wormhole routers). To remedy this predicament, Virtual Channel (VC) flow control [57] assigns multiple virtual paths (each with its own associated buffer queue) to the same physical channel. It has been shown that VC routers can increase throughput by up to 40% over wormhole routers without VCs [54]. As a side bonus, virtual channels can also help with deadlock avoidance [58]. The work in this report assumes, without loss of generality, the use of VC-based wormhole flow control, which suits the low buffer requirements of NoC routers.

The way virtual channels – and hence buffers – are organized within a router is also instrumental in optimizing performance. The number of VCs per physical channel and the VC depth are two parameters that form an elaborate interplay between buffer utilization, throughput and latency. Researchers in the macro-network field have identified the decisive role of virtual channel organization in overall system performance [59,60]. Detailed studies of the relation between virtual channels and network latency indicate that for low traffic intensity, a small number of VCs can suffice. In high traffic rates, however, increasing the number of VCs is a more effective way of improving performance than simply increasing the buffer depth [60]. Under light network traffic, the number of packets traveling through a router is small enough to be accommodated by a limited number of VCs; increasing the number of VCs yields no tangible benefits. Under high traffic, many packets are contenting for router resources; increasing VC depth will not alleviate this contention because of Head-of-Line (HoL) blocking. Increasing the number of VCs, though, will allow more packets to share the physical channels. This dichotomy in VC organization implies that routers with fixed buffer structures will either be underutilized or will underperform under certain traffic conditions, as illustrated in the examples of Fig. 3.1 This figure highlights the weaknesses of statically-partitioned buffers.

Since buffer resources come at a premium in resource-constrained NoC environments (they consume valuable power and silicon real-estate), it is imperative to limit the buffer size to a minimum without severely affecting performance. This objective function can only be achieved through the use of efficient management techniques which optimize buffer utilization. Since size and organization are design-time decisions, they cannot be dynamically changed during operation based on observed traffic patterns. However, the use of a carefully designed buffer controller can significantly affect the efficiency of storing and forwarding of the flits. Therefore, the throughput of a switch can be maximized through dynamic and real-time throttling of buffer resources.

Given the aforementioned significance of the NoC buffers in the area, power and performance triptych, we thereby introduce ViChaR[1]: a dynamic Virtual Channel Regulator, which dispenses VCs according to network traffic. The ViChaR module

[1] The name ViChaR was intentionally chosen to echo the word Vicar, who is someone acting as a substitute or agent for a superior.

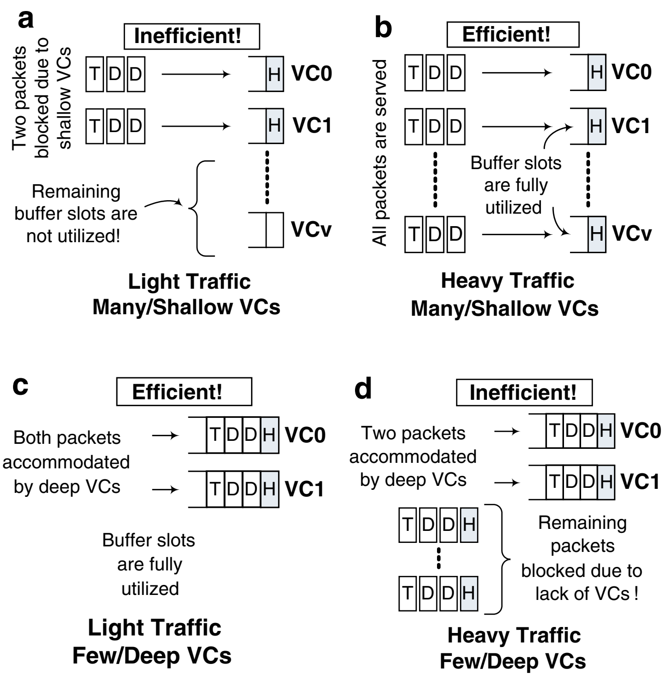

a
Inefficient!
Two packets blocked due to shallow VCs
T D D
H VC0
T D D
H VC1
Remaining buffer slots are not utilized!
VCv
Light Traffic Many/Shallow VCs
b
Efficient!
All packets are served
T D D
H VC0
H VC1
Buffer slots are fully utilized
H VCv
Heavy Traffic Many/Shallow VCs
c
Efficient!
Both packets accommodated by deep VCs
T D D H VC0
T D D H VC1
Buffer slots are fully utilized
Light Traffic Few/Deep VCs
d
Inefficient!
Two packets accommodated by deep VCs
T D D H VC0
T D D H VC1
Remaining packets blocked due to lack of VCs !
Heavy Traffic Few/Deep VCs

Fig. 3.1 Limitations of a statically assigned buffer organization (H = head flit, D = data/middle flit, T = tail flit). (**a**) Light traffic many/shallow VCs. (**b**) Heavy traffic many/shallow VCs. (**c**) Light traffic few/deep VCs. (**d**) Heavy traffic few/deep VCs

is a very compact unit operating at the granularity of one router input/output port; therefore, a conventional 5-port NoC router would employ five such units to oversee buffer management.

ViChaR's operation revolves around two intertwined concepts which constitute the *two fundamental contributions* of this work:

(1) ViChaR Uses a Unified Buffer Structure (UBS), instead of individual and statically partitioned First-In-First-Out (FIFO) buffers. While the unified buffer concept is not new, in this work we are revisiting the concept within the confinements of the strict resource limitations of on-chip networks. This is the first attempt to incorporate a unified buffer management in NoC routers. The new flavor in our endeavor stems from a fundamentally different implementation approach: we introduce a novel, table-based design that provides single-clock operation without incurring prohibitive overhead. Most importantly though, it enables the use of a flexible and dynamically varying virtual channel management scheme, thereby replacing the conventional static resource allocation.
(2) ViChaR provides each individual router port with a variable number of VCs, each of which is dispensed dynamically according to network traffic conditions. This translates into fewer but deeper VCs under light traffic, and more but shallower VCs under heavy traffic. This attribute successfully marries two

contradicting buffer organizations, which are impossible to combine in conventional, statically-allocated buffers. Furthermore, ViChaR's dynamic allocation scheme ensures a smooth continuum between these two extremes (few/deeper VCs versus more/shallower VCs) as the network intensity fluctuates.

3.2 Related Work in Buffer Design

As previously mentioned, interest in packet-based on-chip networks has rapidly gained momentum over the last few years, and analysis and optimization of on-chip interconnect architectures have garnered great attention. In this sub-section, we focus solely on buffer related aspects. Within the realm of on-chip buffer design, both size and organization have been shown to be directly related to network performance [52]. Buffer sizing in particular has been investigated in [52,53]. However, these papers adopt a static approach, where optimal buffer sizes are predetermined at design-time based on a detailed analysis of application-specific traffic patterns. The sizing is optimal for only one particular application and one hardware mapping. However, a technique to alter the buffer organization dynamically at run-time is more desirable for a general purpose and reconfigurable SoC executing different workloads. A dynamic scheme would maximize utilization regardless of the traffic type in the NoC.

Centralized buffer organizations have been studied extensively in the macro-network realm, but the solutions proposed are not amenable to resource constrained on-chip implementations. In particular, a unified and dynamically-allocated buffer structure was originally presented in [61] in the form of the Dynamically Allocated Multi-Queue (DAMQ) buffer. However, whereas the DAMQ architecture was part of a single-chip communication coprocessor for multi-computer systems, the proposed implementation in this report is aimed at area- and power-constrained, ultra-low latency on-chip communication. This profoundly affected our design considerations as follows:

(1) The DAMQ used a fixed number of queues (i.e., virtual channels) per input port. Specifically, four queues were used, one for each of three output ports and a local processor interface. Consequently, all packets in the same queue had to obey the FIFO order, i.e., all packets in the same queue could still get stuck behind a blocked packet at the head of the queue.

(2) The control logic of the DAMQ buffer was very complex, relying on a system of linked lists to organize the data path. These linked lists were stored in pointer registers which had to be updated constantly. This caused a three-cycle delay for every flit arrival/departure, mainly because data had to be moved between pointer registers, and a so-called "free list" had to be updated (a linked list keeping track of available buffer slots) [62]. This three-cycle delay – while acceptable for inter-chip communication – would prove intolerable in an on-chip router.

The DAMQ project spawned a few other designs, which aimed to simplify the hardware implementation and lower overall complexity. Two notable examples of these designs were the DAMQ with self-compacting buffers [63] and the Fully Connected Circular Buffer (FC-CB) [64]. Both designs have less overhead than the linked-list approach of [61] by employing registers, which selectively shift some flits inside the buffer to enable all flits of one VC to occupy a contiguous buffer space. The FC-CB design [64] improves on [63] by using a circular structure, which shifts in only one direction and ensures that any flit will shift by at most one position each cycle. However, the FC-CB has two main disadvantages when applied to an on-chip network. First, being fully connected, it requires a $P^2 \times P$ crossbar instead of the regular $P \times P$ crossbar for a P-input switch. Such large and power-hungry crossbars are unattractive for on-chip routers. Second, the circular shifter allows an incoming flit to be placed anywhere in the buffer and requires selective shifting of some parts of the buffer while leaving the rest of the buffer undisturbed. This functionality inflicts considerable increases in latency, area and power over a simple, non-shifting buffer implementation, like the proposed ViChaR design. The overhead is due to the large MUXes, which are required between each buffer slot to enable both shifting and direct input.

The circular-shift buffer of the FC-CB was implemented in Verilog HDL and synthesized in 90 nm commercial TSMC libraries to assess its viability in on-chip designs. The circular buffer implementation of the FC-CB increases the datapath delay by 26% compared to ViChaR's stationary (i.e., non-shifting) buffer. Increases in datapath delay may affect the pipeline period in deeply pipelined router designs; a longer period will adversely affect throughput. Moreover, the FC-CB's large MUXes incur an increase of approximately 18% in buffer area. More importantly, though, the continuous shifting of the FC-CB buffer every clock cycle (assuming continuous incoming traffic) increases the dynamic power budget by 66%. Obviously, this overhead renders the FC-CB implementation unattractive for on-chip applications. Finally, the FC-CB still works with a fixed number of VCs, just like the DAMQ design. In this report, we will show that a dynamically variable number of VCs optimizes performance.

The notion of dynamically allocating VC resources based on traffic conditions was presented in [65], through the VCDAMQ and DAMQ-with-recruit-registers (DAMQWR) implementations. However, both designs were coupled to DAMQ underpinnings; hence, they employed the linked-list approach of the original DAMQ, which is too costly for an on-chip network. Nevertheless, the work of [65] highlighted the significance of dynamic allocation of buffer resources, which forms the premise of the design proposed in this report.

Finally, the Chaos router [66] and BLAM routing algorithm [67] provide an alternative technique to saving buffer space. They employ packet misrouting, instead of storage, under heavy load. However, randomized (non-minimal) routing may make it harder to meet strict latency guarantees required in many NoCs (e.g., multimedia SoCs). Moreover, these schemes do not support dynamic VC allocation to handle fluctuating traffic.

3.3 The Proposed Dynamic Virtual Channel Regulator (ViChaR)

So far, as a result of scarce area and power resources and ultra-low latency requirements, on-chip routers have relied on very simple buffer structures. In the case of virtual channel-based NoC routers, these structures consist of a specified number of FIFO buffers per input port, with each FIFO corresponding to a virtual channel. This is illustrated in Fig. 3.2 Such organization amounts to a static partitioning of buffer resources. Hence, each input port of an NoC router has *v* virtual channels, each of which has a dedicated *k*-flit FIFO buffer. Current on-chip routers have small buffers to minimize their overhead; *v* and *k* are usually much smaller than in macro networks [48]. The necessity for very low latency dictates the use of a parallel FIFO implementation, as shown in the bottom right of Fig. 3.2 As opposed to a serial FIFO implementation [68], the parallel flavor eliminates the need for a flit to traverse all slots in a pipelined manner before exiting the buffer [68]. This fine-grained control requires more complex logic, which relies on read and write pointers to maintain the FIFO order. Given the small sizes of on-chip buffers, though, the inclusion of a parallel FIFO implementation is by no means prohibitive. The buffers within an NoC router can be implemented as either registers or SRAM/DRAM memory [69,70]. However, given the relatively small buffer sizes employed, it is more reasonable to use small registers as buffers to avoid the address decoding/encoding latencies of big memories and the access latencies associated with global bitlines/wordlines [69]. To this extent, the NoC buffers in this report were implemented as registers.

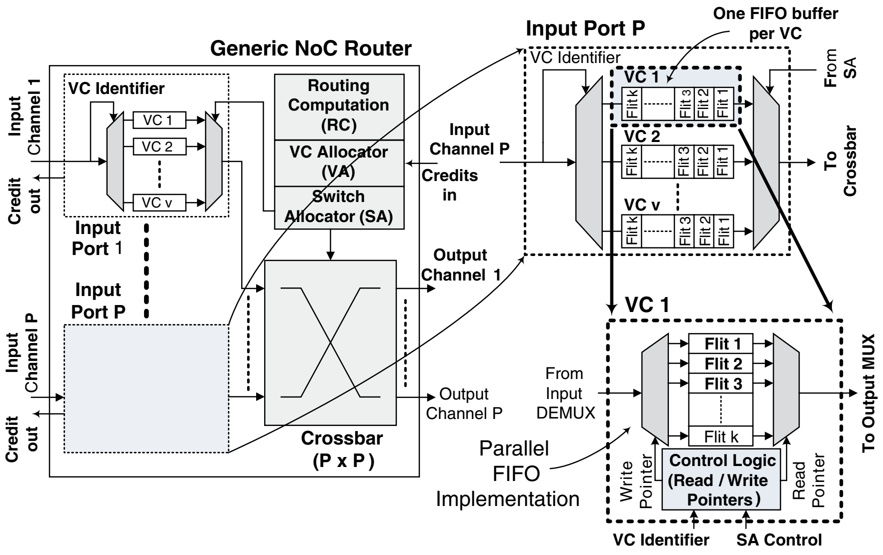


Fig. 3.2 A generic NoC router architecture and buffer structure

FIFO buffers in statically assigned buffer structures have two inherent disadvantages. First, a packet at the head of a VC whose designated output port is busy will block all subsequent packets in that VC from being transmitted (assuming non-atomic buffer allocation) even if their designated output ports are free. This Head-of-Line (HoL) blocking can severely affect network performance in congested conditions, similar to the previously discussed DAMQ. This scenario is illustrated at the top of Fig. 3.3 Second, if only part of a packet occupies a VC buffer at a given time, then any vacant slots in that buffer cannot be reassigned to a new packet for as long as that VC is reserved by the partial packet to avoid packet/message mixing. Thus, a VC buffer may only be occupied by a single header flit because the remaining flits happen to be blocked in preceding routers due to congestion. In such a scenario, the remaining free slots in the buffer cannot be assigned to other packets until the tail flit of the current packet releases the VC. This attribute of FIFO buffers can lead to substantial under-utilization of the buffers, as shown at the bottom of Fig. 3.3 and cripple network performance.

Figure 3.4 illustrates the buffer organization of a conventional NoC router (left) and our proposed alterations (right). The crux of ViChaR is composed of two main components: (1) the ***Unified Buffer Structure (UBS)***, shown in Fig. 3.4, and (2) the associated control logic, called ***Unified Control Logic (UCL)***.

Figure 3.4 shows only one of the five sub-modules of UCL, the *Arriving/Departing Flit Pointer Logic*. This sub-module constitutes the interface between the UBS and the UCL; the UCL controls the unified buffer (UBS) through the Arriving/Departing Flit Pointer Logic module. A top-level block diagram of the entire ViChaR architecture is shown in Fig. 3.6. This figure illustrates all five of the UCL sub-modules: (1) the *Arriving/Departing Flit Pointers Logic*, (2) the *Slot Availability Tracker*, (3) the *VC Availability Tracker*, (4) the *VC Control Table*, and (5) the *Token (VC) Dispenser*. The operation of each component and the interaction

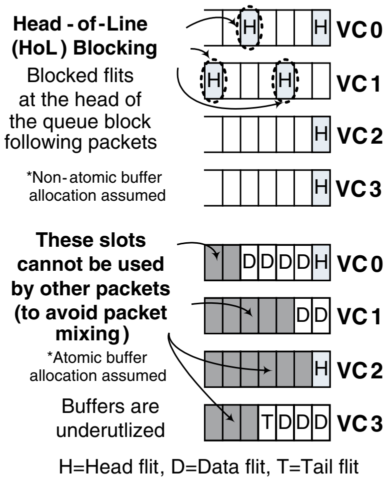

Head - of - Line (HoL) Blocking
Blocked flits at the head of the queue block following packets
*Non - atomic buffer allocation assumed
VC 0
VC 1
VC 2
VC 3
H
These slots cannot be used by other packets (to avoid packet mixing)
*Atomic buffer allocation assumed
Buffers are underutlized
D
T
H=Head flit, D=Data flit, T=Tail flit

Fig. 3.3 Limitations of existing FIFO buffers

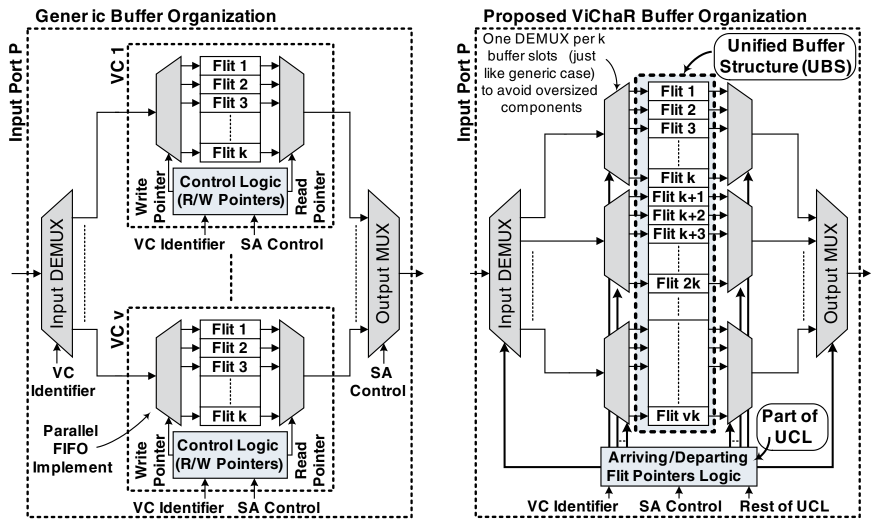

Gener ic Buffer Organization
Input Port P
VC 1
Flit 1
Flit 2
Flit 3
Flit k
Write Pointer
Control Logic (R/W Pointers)
Read Pointer
VC Identifier SA Control
Input DEMUX
Output MUX
VC v
VC Identifier
SA Control
Parallel FIFO Implement
Proposed ViChaR Buffer Organization
One DEMUX per k buffer slots (just like generic case) to avoid oversized components
Unified Buffer Structure (UBS)
Flit k+1
Flit k+2
Flit k+3
Flit 2k
Flit vk
Part of UCL
Arriving/Departing Flit Pointers Logic
VC Identifier SA Control Rest of UCL

Fig. 3.4 The proposed ViChaR architecture

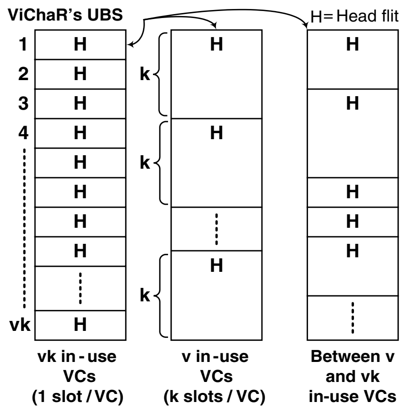

ViChaR's UBS
H= Head flit
1
2
3
4
vk
H
k
vk in - use VCs (1 slot / VC)
v in - use VCs (k slots / VC)
Between v and vk in-use VCs

Fig. 3.5 Possible VC configurations in ViChaR

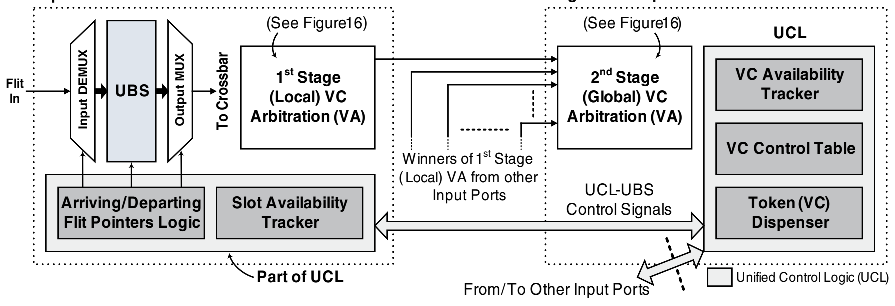

Input Port P
Flit In
Input DEMUX
UBS
Output MUX
To Crossbar
(See Figure16)
1st Stage (Local) VC Arbitration (VA)
Logic For Output Port P
2nd Stage (Global) VC Arbitration (VA)
Winners of 1st Stage (Local) VA from other Input Ports
UCL
VC Availability Tracker
VC Control Table
Token (VC) Dispenser
Arriving/Departing Flit Pointers Logic
Slot Availability Tracker
UCL-UBS Control Signals
Part of UCL
From/To Other Input Ports
Unified Control Logic (UCL)

Fig. 3.6 ViChaR block diagram (only one of P ports shown here)

between the UBS and its controlling entity (the UCL) are described in detail in Section 3.3.2. All five modules function independently and in parallel, which is of critical importance to the ultra-low latency requirements of the router. The UCL components work in tandem with the unified buffer (UBS), providing dynamic allocation of both virtual channels and their associated buffer depth. As illustrated in Fig. 3.6, the two main ViChaR components (UBS and UCL) are logically separated into two groups: the unified buffer (UBS) and two of the five UCL modules (the Arriving/Departing Flit Pointers Logic and the Slot Availability Tracker) are situated at the input side of the router (i.e., to accept all incoming flits), while the remaining modules of the control logic (UCL) are responsible for the VC arbitration of all flits destined to a particular output port. Based on incoming traffic and information from the Slot and VC Availability Trackers, the Token (VC) Dispenser grants VC IDs to new packets accordingly. The VC Control Table is the central hub of ViChaR's operation, keeping track of all in-use VCs and a detailed status of the unified buffer (UBS). When flits arrive and/or depart, the Arriving/Departing Flit Pointers Logic controls the UBS's MUXes and DEMUXes in accordance with the VC Control Table.

It is important to realize that the UBS is physically identical to the generic buffer structure: the v independent k-flit FIFO buffers of a traditional implementation are simply *logically* grouped in a single vk-flit entity (the UBS in Fig. 3.4). Hence, other than the control logic, there is no additional hardware complexity, since the vk-flit UBS is NOT a large, monolithic structure; it groups the existing buffers together, and it is only through the use of its control mechanism (the UCL) that the buffers appear as a *logically unified structure*. As shown in Fig. 3.4, UBS retains the same number of MUXes/DEMUXes as the generic implementation, i.e., one MUX/DEMUX per k flits, to avoid large (and hence slower) components.

3.3.1 Variable Number of Virtual Channels

Whereas a conventional NoC router can support only a fixed, statically assigned number of VCs per input port (namely v, as shown in Fig. 3.4), the ViChaR architecture can have a variable number of assigned VCs, based on network conditions. *ViChaR assigns at most one packet to each VC* so as to enable fine flow control granularity; on the contrary, the sharing of a single VC by multiple packets can lead to situations where a blocked packet impedes the progress of another packet which happens to use the same VC (known as HoL blocking, as described in Section 3.3). A vk-flit ViChaR structure can support anywhere between v VCs (when each VC occupies the maximum of k flits) and vk VCs (when each VC occupies the minimum of 1 flit) at any given time under full load. This variability in the number of in-use VCs is illustrated in Fig. 3.5. To aid understanding, each VC in Fig. 3.5 is shown to occupy a contiguous space in the buffer (UBS); in reality, however, this may not be the case because the UBS allows the VCs to include non-consecutive buffer slots (this fact will be explained in more detail in Section 3.3.2). Hence, the

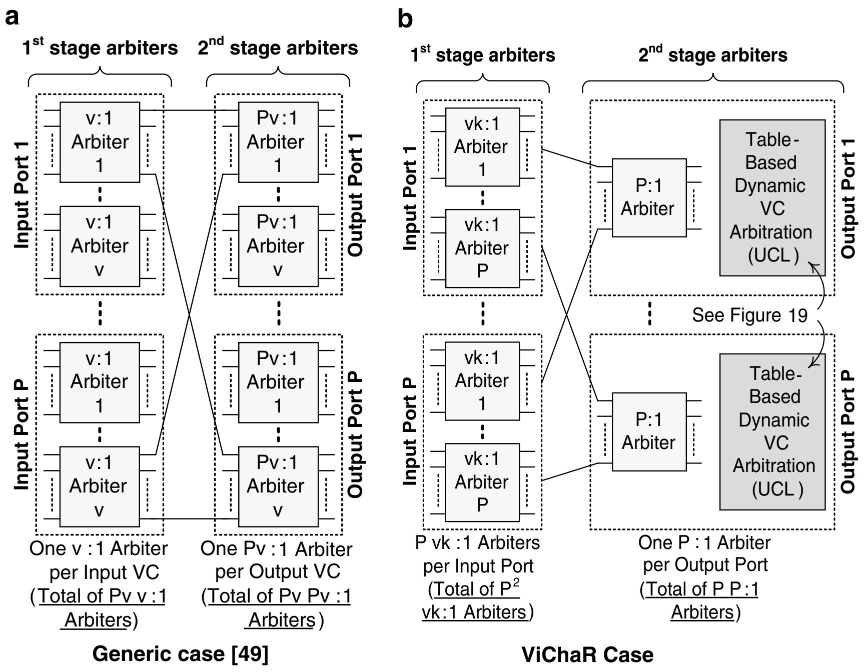

a
1st stage arbiters
2nd stage arbiters
Input Port 1
Input Port P
Output Port 1
Output Port P
v:1 Arbiter 1
v:1 Arbiter v
Pv:1 Arbiter 1
Pv:1 Arbiter v
One v : 1 Arbiter per Input VC (Total of Pv v : 1 Arbiters)
One Pv : 1 Arbiter per Output VC (Total of Pv Pv : 1 Arbiters)
Generic case [49]
b
1st stage arbiters
2nd stage arbiters
vk:1 Arbiter 1
vk:1 Arbiter P
P:1 Arbiter
Table-Based Dynamic VC Arbitration (UCL)
See Figure 19
P vk : 1 Arbiters per Input Port (Total of P2 vk:1 Arbiters)
One P : 1 Arbiter per Output Port (Total of P P : 1 Arbiters)
ViChaR Case

Fig. 3.7 Virtual channel arbitration (VA). (**a**) Generic case [49]. (**b**) ViChaR case

system can support a variable number of in-flight packets per port, dynamically allocating new VCs when network conditions dictate it. Dynamic variability in in-flight messages can increase throughput under heavy traffic.

As a result of its unified buffer and dynamic behavior, the ViChaR structure alters the Virtual channel Allocation (VA) logic of the router. Since the router function may return multiple output VCs restricted to a single physical channel [49], two arbitration stages are required in both the generic and ViChaR cases, as shown in Fig. 3.7. In the generic case, the first stage reduces the number of requests from each input VC to one (this ensures the request of a single VC at a particular output port by each input VC). Subsequently, the winning request from each input VC proceeds to the second arbitration stage. Details of the VA operation are omitted for brevity, but can be found in [49].

In the proposed ViChaR architecture, VA takes a different approach due to the dynamic VC allocation scheme: the first arbitration stage reduces the number of requests for *a particular output port* to one request per input port. The generic router (Fig. 3.7a) requires v:1 arbiters, since the number of VCs supported is fixed to v. ViChaR, on the other hand, supports anywhere between v and vk VCs per port at any given time. To accommodate the worst case scenario (i.e., vk in-flight VCs), ViChaR needs larger vk:1 arbiters in stage 1 of the allocation (Fig. 3.7b). The second arbitration stage in ViChaR produces *a winner for each output port among all the competing input ports*. Therefore, while the proposed ViChaR architecture uses

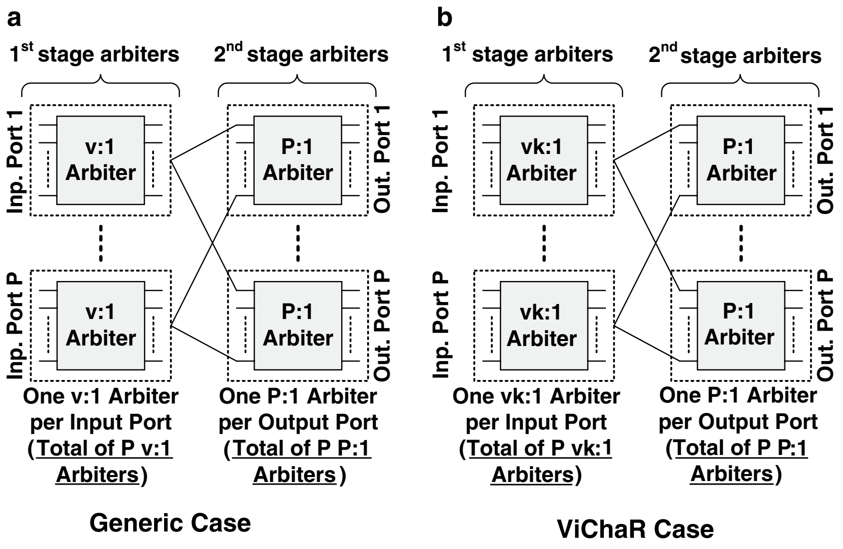


Fig. 3.8 Switch allocation (SA). (**a**) Generic case. (**b**) ViChaR case

larger Stage 1 arbiters (*vk*:1 vs. *v*:1), it uses much smaller and fewer Stage 2 arbiters. The reason for the simplified second stage is that ViChaR dynamically allocates VCs as needed, instead of accepting requests for specific VCs (which would necessitate one arbiter per output VC, just like the generic case). It is this attribute that helps the ViChaR implementation incur only a slight increase in power consumption (and even achieve a small area decrease), compared to a generic architecture, as will be shown shortly.

The variable number of VCs supported by ViChaR also necessitates bigger arbiters in the first stage of Switch Allocation (SA), as shown in Fig. 3.8. Similar to VA, switch allocation is performed in two stages. The first stage accounts for the sharing of a single port by a number of VCs. Again, ViChaR needs larger *vk*:1 arbiters. The second stage arbitrates between the winning requests from each input port (i.e., *P* ports) for each output port; thus, it is the same for both architectures. The ViChaR overhead due to the bigger stage-1 SA arbiters (illustrated in Table 1's detailed breakdown) is almost fully amortized by the bigger savings resulting from the smaller VA stage discussed previously.

To analyze the area and power overhead, NoC routers with (a) a generic buffer and (b) the proposed ViChaR buffer were implemented in structural Register-Transfer Level (RTL) Verilog and then synthesized in Synopsys Design Compiler using a TSMC 90 nm standard cell library. The resulting designs operate at a supply voltage of 1 V and a clock frequency of 500 MHz. The routers have 5 input ports (i.e., $P=5$), 4 VCs per input port (i.e., $v=4$), each VC is four-flit deep (i.e., $k=4$), and each flit is 128 bits long. Both area and power estimates were extracted from the synthesized router implementations. A comparison of the area and power overhead of the two schemes is shown in Table 1. Note that both routers have equal buffer space ($vk=16$ buffer slots per input port) for fairness. It is evident that while ViChaR incurs an overhead in terms of control logic and switch allocation (SA), this overhead is over-compensated (in terms of area) by a larger reduction in the VA logic. Thus, the ViChaR model provides area savings of around 4%. In terms of power, ViChaR consumes slightly more power (1.75%). This power increase, however, is negligible compared to the performance benefits of ViChaR, as will be demonstrated in Section 3.4.

3.3.2 ViChaR Component Analysis

The key challenges in designing ViChaR were to *avoid* (a) deepening the router's pipeline, and (b) decreasing the operating frequency. To circumvent the multi-cycle delay induced by a linked-list approach [61] to ViChaR, we opted instead for a *table-based* approach, as illustrated in Fig. 3.10. This logic is required for each output port in the router. Following is a break-down of the control logic (UCL) sub-modules of the proposed ViChaR architecture:

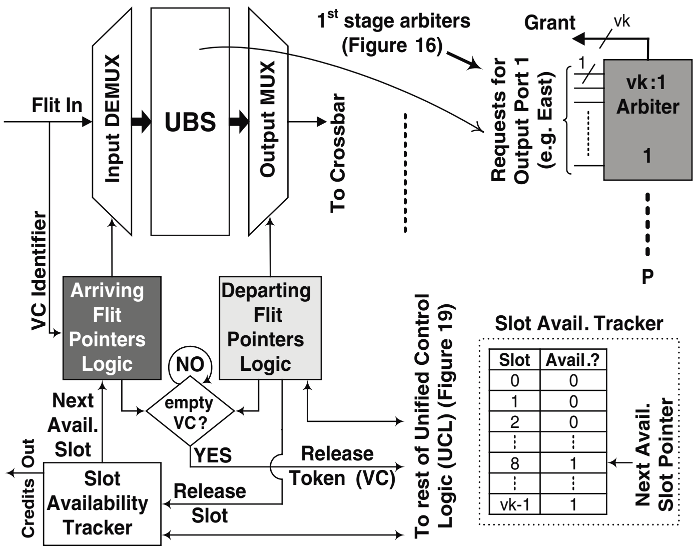
VC Control Table The VC Control Table (see Fig. 3.10) forms the core of the control logic of ViChaR. It is a compact table, holding the slot IDs of all flits currently in the buffers, which are requesting the particular output port (e.g., West). Note that since the number of buffer slots in on-chip routers is resource-constrained, the size of the table is minimal, as demonstrated by the low overhead in the control logic in Table 3.1. The VC Control Table is organized by VC ID, with each VC having room for at most a single packet. Without loss of generality, in this work we assumed a packet to consist of four flits: a Head flit, two Data (middle) flits, and a

Table 3.1 Area and power overhead of the ViChaR architecture. The results in this table assume equal-size buffers for both router designs. However, ViChaR's efficient buffer management scheme allows for a 50% decrease in buffer size with no performance degradation (see Section 3.4). In such a case, area and power is reduced by 30% and 34%, respectively, over the whole router

Component (one input port)	Area (in μm^2)	Power (in mW)
ViChaR table-based control logic	12,961.16	5.36
ViChaR buffer slots (16 slots)	54,809.44	15.36
ViChaR VA logic	27,613.54	8.82
ViChaR SA logic	6,514.90	2.06
TOTAL for ViChaR architecture	101,899.04	31.60
Generic control logic	10,379.92	5.12
Generic buffer slots (16 slots)	54,809.44	15.36
Generic VA logic	38,958.80	9.94
Generic SA logic	2,032.93	0.64
TOTAL for generic architecture	106,181.09	31.06
ViChaR overhead/savings	−4,282.05	+0.54
	4.03% savings	1.74% overhead

Tail flit. The packet size is assumed to be constant, but the table can trivially be changed to accommodate a variable-sized packet protocol. As seen in the VC Control Table box of Fig. 3.10 (right-hand side), the VCs can include non-consecutive buffer slots (e.g., VC1 comprises of slots 2, 4, 6 and 7) of the South input port (i.e., flits arriving from the South). This attribute allows full-flexibility in buffer utilization and avoids the issues encountered in statically-allocated buffers. VC3 only occupies one slot (10) in Fig. 3.10. In a static buffer, 3 additional slots would have to be reserved for the remaining flits of VC3; those slots would remain unused if the remaining flits happened to be blocked in previous routers. Instead, in ViChaR those slots can be used by other VCs, thus maximizing the buffer utilization. Furthermore, the use of a table-based controller makes the management of a variable number of VCs very easy: non-used VCs are simply NULLed out in the VC Control Table (e.g., VC4 in Fig. 3.10).

Arriving/Departing Flit Pointers Logic The Flit Pointers Logic directly controls the Input and Output MUXes/ DEMUXes of the unified buffer (UBS), as illustrated in Fig. 3.9, and is directly linked to the VC Control Table module. Once a flit departs, its location in the VC Control Table is invalidated by asserting a NULL bit. There is a set of such pointers for each VC in the table. However, the overhead is minimal due to the simplicity of the pointer logic; both Departing and Arriving Flit Pointers are implemented in combinational logic and simply have to observe the non-NULL locations in their VC. For example, the Departing Flit pointer points at the first non-NULL location (in its particular VC) starting from the left of the table, as shown on the right side of Fig. 3.10 for VC2 (in the VC Control Table box). If all the entries in a single row of the VC Control Table are NULL, then the VC must be empty; thus, the pointer logic releases the VC by notifying the VC Availability



Flit In
Input DEMUX
UBS
Output MUX
To Crossbar
1st stage arbiters (Figure 16)
Grant
vk
1
Requests for Output Port 1 (e.g. East)
vk:1 Arbiter
1
P
VC Identifier
Arriving Flit Pointers Logic
Departing Flit Pointers Logic
NO
empty VC?
YES
Next Avail. Slot
Credits Out
Slot Availability Tracker
Release Slot
Release Token (VC)
To rest of Unified Control Logic (UCL) (Figure 19)
Slot Avail. Tracker
Next Avail. Slot Pointer

Slot	Avail.?
0	0
1	0
2	0
⋮	⋮
8	1
⋮	⋮
vk-1	1

Fig. 3.9 The ViChaR UBS architecture (one input port shown)

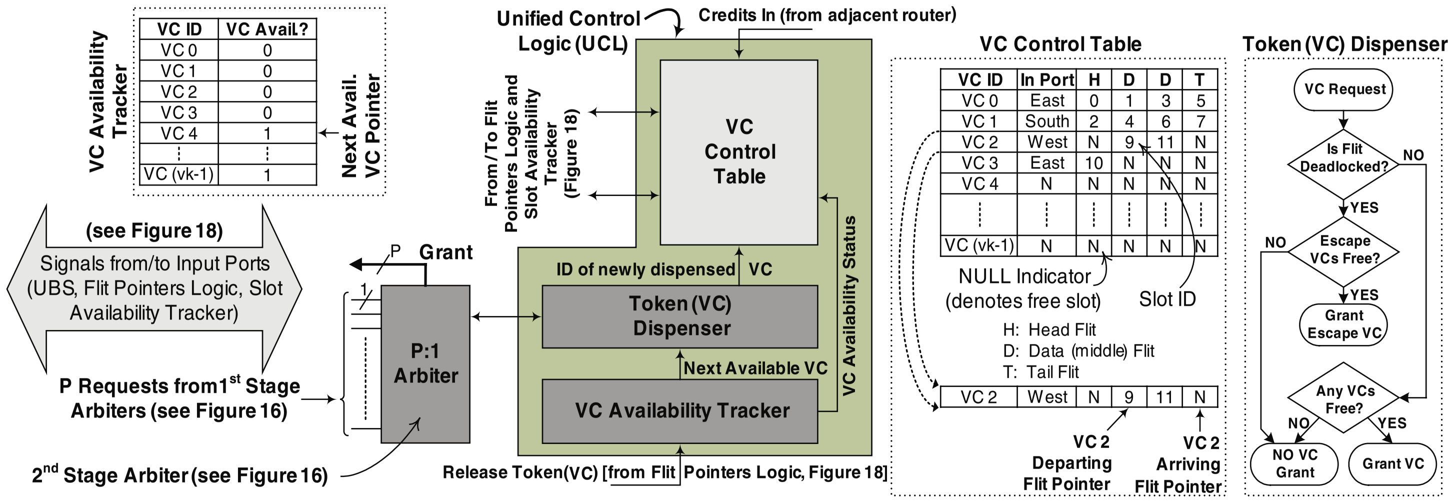

VC Availability Tracker
VC ID | VC Avail?
VC 0 | 0
VC 1 | 0
VC 2 | 0
VC 3 | 0
VC 4 | 1
VC (vk-1) | 1
Next Avail. VC Pointer
(see Figure 18)
Signals from/to Input Ports (UBS, Flit Pointers Logic, Slot Availability Tracker)
P Requests from 1st Stage Arbiters (see Figure 16)
2nd Stage Arbiter (see Figure 16)
P
Grant
1
P:1 Arbiter
Unified Control Logic (UCL)
Credits In (from adjacent router)
From/To Flit Pointers Logic and Slot Availability Tracker (Figure 18)
VC Control Table
ID of newly dispensed VC
Token (VC) Dispenser
Next Available VC
VC Availability Tracker
VC Availability Status
Release Token(VC) [from Flit Pointers Logic, Figure 18]
VC Control Table
VC ID | In Port | H | D | D | T
VC 0 | East | 0 | 1 | 3 | 5
VC 1 | South | 2 | 4 | 6 | 7
VC 2 | West | N | 9 | 11 | N
VC 3 | East | 10 | N | N | N
VC 4 | N | N | N | N | N
VC (vk-1) | N | N | N | N | N
NULL Indicator (denotes free slot)
Slot ID
H: Head Flit
D: Data (middle) Flit
T: Tail Flit
VC 2 | West | N | 9 | 11 | N
VC 2 Departing Flit Pointer
VC 2 Arriving Flit Pointer
Token (VC) Dispenser
VC Request
Is Flit Deadlocked?
NO
YES
Escape VCs Free?
NO
YES
Grant Escape VC
Any VCs Free?
NO
YES
NO VC Grant
Grant VC

Fig. 3.10 ViChaR table-based UCL architecture (logic for one of P ports shown)

Tracker (Release Token signal in Fig. 3.9). When a new flit arrives, the pointer logic guides the flit to the appropriate slot in the unified buffer (UBS), based on the flit's VC ID and information from the Slot Availability Tracker. Finally, newly arrived header flits in the UBS can request an output VC by first undergoing local (1st stage) arbitration (top right of Fig. 3.9), and then global (2nd stage) arbitration (bottom left of Fig. 3.10).

VC and Slot Availability Trackers The VC Availability Tracker simply keeps track of all the VCs in the VC Control Table that are not used. The Token (VC) Dispenser dynamically assigns VCs to new incoming packets based on information provided by the VC Availability Tracker. Similarly, the Slot Availability Tracker keeps track of all the UBS slots that are not in use. When a new flit arrives, it is stored into a slot indicated by the Slot Availability Tracker. The VC and Slot Availability Trackers are functionally identical. They consist of a small table, as shown at the bottom right of Fig. 3.9 (Slot Availability Tracker) and the top left of Fig. 3.10 (VC Availability Tracker). Each row of the table corresponds to one VC ID (in the VC Availability Tracker) or one buffer slot (in the Slot Availability Tracker). For each entry in the table, one bit indicates that the VC/Slot is available (logic 1) or occupied (logic 0). Both trackers have a pointer that points to the top-most available entry. If all VCs are occupied (i.e., all-zero table in the VC Availability Tracker), the Token (VC) Dispenser stops granting new VCs to requesting packets. Similarly, an all-zero table in the Slot Availability Tracker implies a full buffer (UBS); this is reflected in the credit information sent to adjacent routers. The functionality of the VC/Slot Availability Trackers is implemented in combinational logic, similar to the Flit Pointers Logic described above.

Token (VC) Dispenser The Token (VC) Dispenser interfaces with the P:1 Arbiter and is responsible for dispensing free VCs to requesting packets. VCs here are like tokens; they are granted to new packets and then returned to the dispenser upon release. The flow diagram of the Dispenser's operation is illustrated on the right-hand side of Fig. 3.10. Based on information provided by the VC Availability Tracker, the Token Dispenser decides whether to grant a VC or not. The VC dispenser keeps checking for possible deadlock situations among the in-use VCs. Deadlocks may occur in networks which employ adaptive routing schemes. If a pre-specified time threshold is exceeded, the Token Dispenser can channel an existing VC into one of the escape VCs to break the deadlock. As a proof of concept of ViChaR's functionality, the experiments in this report use deterministic (XY) routing, which is inherently deadlock-free. However, ViChaR was designed to operate under adaptive routing schemes as well. Therefore, the Token (VC) Dispenser needs to account for possible deadlock situations. Toward that extent, a number of VCs can be designated as "escape", or "drain" channels to provide deadlock recovery in adaptive routing algorithms (escape channels employ a deterministic routing algorithm to break the deadlock) [71]. The Dispenser needs to switch deadlocked flits into these escape channels, if there is a need. One experiment in Section 3.4.2 validates the effectiveness of this technique under adaptive routing.

Assuming that no deadlock situation exists, the Token Dispenser can proceed with its normal operation. The Dispenser grants new VCs on a First-Come-First-Served (FCFS) basis; if a new header flit wins the VC arbitration and the VC Availability Tracker indicates that a free VC is available, then the new packet will be granted a new VC. The Dispenser does not give priority to flits of existing VCs. In principle, a more elaborate mechanism could be used for dispensing new VCs, which would monitor on-going traffic and reach a decision based on some quantitative metric or prior traffic history. However, given the highly restrictive objective function of minimal area, power and latency budgets in the design of on-chip networks, such complex monitoring mechanisms were deemed infeasible. After all, ViChaR was architected to operate within one clock cycle. The use of an FCFS scheme in the Token Dispenser turns out to be very efficient at maximizing performance. The Dispenser is able to self-throttle the dispensing of new VCs based on traffic conditions: if more packets request a channel (high traffic) more VCs are dispensed; if fewer packets are present (low traffic) fewer VCs are granted and more buffer depth is allotted to existing VCs.

ViChaR's Effect on the Router Pipeline The control logic (UCL) of ViChaR was designed in such a way as to decouple the operation of the sub-modules from each other. Thus, sub-modules are kept compact and can all operate in parallel, hence completing the entire operation in a single clock cycle. This is a significant improvement over the three-clock cycle delay of [61]. Figure 3.11 shows the pipeline stages of both a generic and the ViChaR router pipelines. As previously mentioned, the ViChaR architecture modifies the VA and SA stages (stages 2 and 3 in Fig. 3.11). The dark-colored boxes indicate the components modified/added in the ViChaR structure as compared to the generic case. As shown in the figure, the additional hardware operates in parallel without affecting the critical path of the router. This fact is also verified by our studies of the critical path delays of all

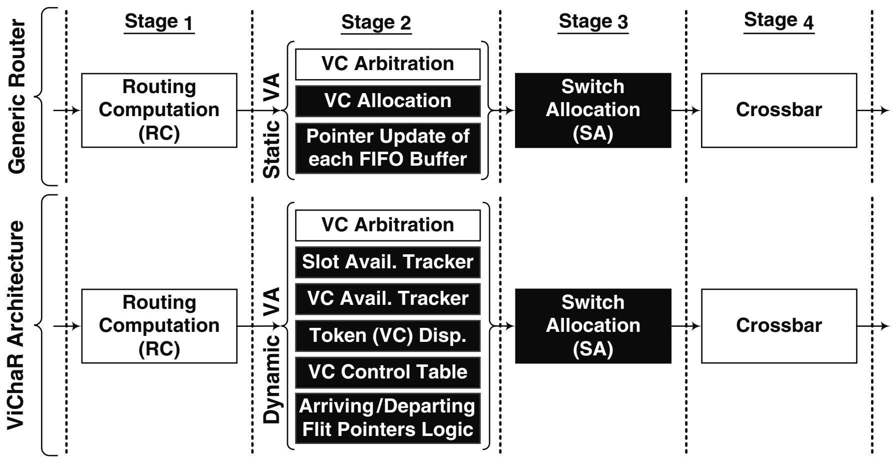


Fig. 3.11 Generic and ViChaR NoC router pipelines

major components of the router architectures (extracted from the synthesized designs). In both cases, the bottleneck that determines the minimum clock period is the arbitration logic (for the VA and SA stages, as shown in Figs. 3.7 and 3.8, respectively). All the components of the ViChaR router remain within the slack provided by the slower arbiters. Hence, the ViChaR architecture does not affect the pipeline depth or the clock frequency. Furthermore, since ViChaR does not create any interdependencies between pipeline stages, it can also be used in speculative router architectures which minimize the pipeline length.

3.4 Simulation Results

3.4.1 Simulation Platform

A cycle-accurate on-chip network simulator was used to conduct detailed evaluation of the architectures under discussion. The simulator operates at the granularity of individual architectural components. The simulation test-bench models the pipelined routers and the interconnection links. All simulations were performed in a 64-node (8×8) MESH network with 4-stage pipelined routers. Each router has 5 physical channels (ports) including the PE-router channel. The generic router (shown as "GEN" in results graphs) has a set of 4 virtual channels per port. Each VC holds four 128-bit flits (i.e., a total of 5×4×4=80 buffer slots). The ViChaR router ("ViC" in results graphs) has a 16-flit unified buffer per port (i.e., a total of 5×16=80 buffer slots, just like the generic case). One packet consists of four flits. The simulator keeps injecting messages into the network until 300,000 messages (including 100,000 warm-up messages) are ejected. Two network traffic patterns were investigated: (1) Uniform Random (UR), where a node injects messages into the network at regular intervals specified by the injection rate, and (2) Self-Similar (SS), which emulates internet and Ethernet traffic. For destination node selection, two distributions were used: (1) Normal Random (NR), and (2) Tornado (TN) [72]. In all cases, except one, deterministic (XY) routing and wormhole switching were employed. One experiment used minimal adaptive routing to evaluate the systems in a deadlock-prone environment. Single link traversal was assumed to complete within one clock cycle at 500 MHz clock frequency. Both dynamic and leakage power estimates were extracted from the synthesized router designs and back-annotated into the network simulator.

3.4.2 Analysis of Results

Our simulation exploration starts with a latency comparison between a conventional, statically assigned buffer architecture and the proposed ViChaR implementation.

We first assume that both designs have equal-sized buffers; specifically, 16-flit buffers per input port (i.e., a total of 80 flits per NoC router).

In the generic design (GEN), the 16 buffer slots are arranged as 4 VCs, each with a 4-flit depth. ViChaR (ViC), on the other hand, can dynamically assign its 16 buffer slots to a variable number of VCs, each with a variable buffer depth. Figure 3.12a, b show the average network latency (in clock cycles) as a function of injection rate (in flits/node/cycle) for Uniform Random (UR) and Self-Similar (SS) traffic patterns, respectively. The graphs include results for both Normal Random (NR) and Tornado (TN) source-destination selection patterns. In all cases, ViChaR substantially outperforms the generic architecture; by 28% (NR) and 24% (TN) on average for Uniform Random traffic, and 25% (NR) and 18% (TN) for Self-Similar traffic. More importantly, though, ViChaR saturates at higher injection rates than the generic case.

Figure 3.12c shows the buffer occupancy at injection rates between 0.25 and 0.35 (i.e., before the onset of saturation). Higher buffer occupancy indicates network blocking. ViChaR is clearly much more efficient at moving flits through the router; the buffer occupancy of a 16-flit/port ViChaR design is considerably lower than an equal-size static configuration. *Buffer occupancy alone, however, is not an indicative metric, since it does not relay any information about network latency*. To validate ViChaR's highly efficient buffer management scheme, its latency at these smaller buffer sizes should also be investigated. To that extent, Fig. 3.12d and Fig. 3.12e depict how the latency of ViChaR at various buffer sizes compares to the latency of the generic architecture with a fixed 16-flit/port buffer size. It is evident from the graphs that *the UBS can achieve similar performance with less than half the buffer size of the generic architecture*. This is of profound importance, since buffers dominate the area and power budgets of NoC routers; reducing the buffer size by 50% will yield significant savings. An alternative way to visualize this phenomenon is illustrated in Fig. 3.12f. This graph shows how the latency of ViChaR at various buffer sizes compares to the latency of the generic architecture with a 16-flit/port buffer size (horizontal dashed line) at an injection rate of 0.25. ViChaR has higher latency only when its buffer size drops below 8 flits per port. On the other hand, Fig. 3.12g shows that decreasing the buffer size in a generic, statically assigned buffer structure always degrades performance.

Following on the very encouraging result that ViChaR can achieve similar performance as a conventional buffer by using only half the buffers, Fig. 3.12h shows the total average power consumption of the 8×8 MESH network for different buffer configurations. For equally sized configurations, ViChaR consumes slightly more power than a conventional buffer structure. At injection rates up to 0.3, ViChaR consumes about 2% more power, corroborating the results of Table 1. At higher injection rates (when the network saturates), excessive switching activity causes this difference to grow a bit more, even though it never exceeds 5%. However, since ViChaR's efficiency allows us to *halve* the buffer resources with no discernible effect on performance, the overall power drops by about 34% (ViC-8 in Fig. 3.12h) for equivalent performance. Similarly, the area occupied by the router decreases by around 30%, based on synthesis results. These decreases can lead to more power- and area-efficient SoCs.

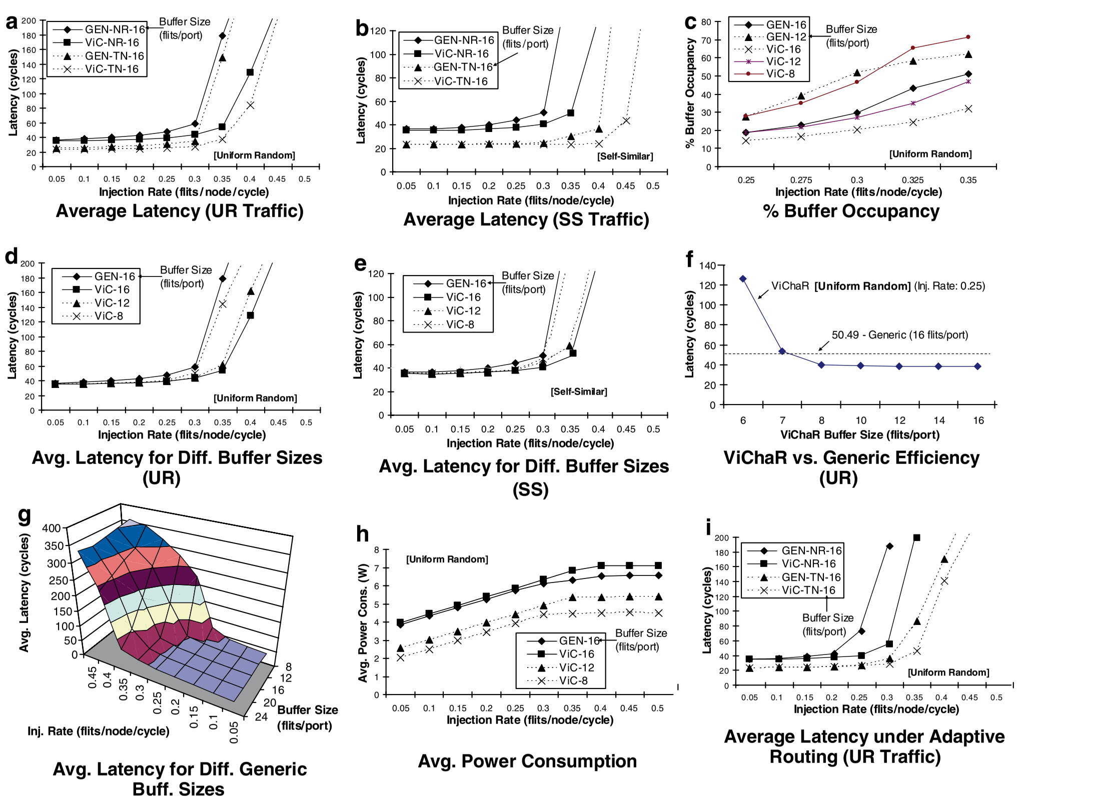


Fig. 3.12 Average latency, % buffer occupancy, and average power consumption simulation results. (**a**) Average latency (UR traffic). (**b**) Average latency (SS traffic). (**c**) % Buffer occupancy. (d) Average latency for different buffer sizes (UR). (**e**) Average latency for different buffer sizes (SS). (**f**) ViChaR vs. generic efficiency (UR). (**g**) Average latency for different generic buffer sizes. (**h**) Average power consumption. (**i**) Average latency under adaptive routing (UR traffic)

Figure 3.12i compares average network latency under minimal adaptive routing to validate ViChaR's effectiveness in handling deadlocks. Escape (drain) channels, which employ deterministic (i.e., deadlock-free) routing, were used in both the generic and ViChaR architectures to break deadlocks. Evidently, ViChaR was able to handle all deadlock situations while significantly outperforming the conventional design.

Figure 3.13a, b present another metric of network performance, namely, throughput (in flits per cycle). These graphs follow the same trend as the latency experiments, with ViChaR clearly outperforming a conventional buffer structure.

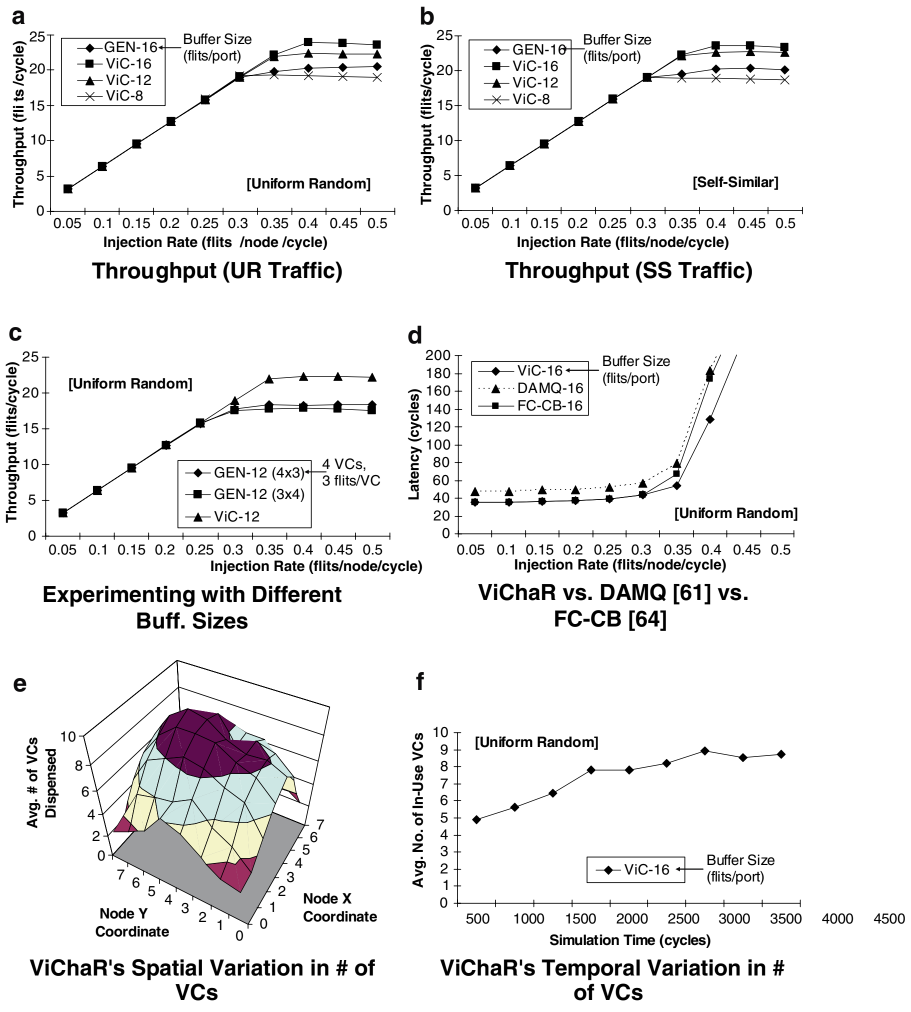


Fig. 3.13 Simulation results demonstrating ViChaR's efficient virtual channel management scheme. (**a**) Throughput (UR traffic). (**b**) Throughput (SS traffic). (**c**) Experimenting with different buffer sizes. (**d**) ViChaR vs. DAMQ [61] vs. FC-CB [64]. (**e**) ViChaR's spatial variation in # of VCs. (**f**) ViChaR's temporal variation in VCs

Figure 3.13c includes the throughput of two different (but of equal size) generic configurations: 4 VCs each with a 3-flit depth, and 3 VCs with a 4-flit depth. The graph indicates that while varying the statically-assigned VC configuration of a generic buffer does affect throughput, its performance still trails that of the dynamically variable design of ViChaR.

In the related work sub-section (Section 3.2), we analyzed in detail why the unified buffers of the DAMQ [61] and FC-CB [64] would underperform compared to ViChaR's dynamic design. Both the DAMQ and FC-CB structures were implemented and incorporated into our cycle-accurate simulator. Figure 3.13d shows how all designs fare against each other. DAMQ loses out because of its 3-cycle buffer delay, and its fixed number of VCs, as previously explained. For a fair comparison, we assumed that the FC-CB design completes its buffer management procedure in one clock cycle (just like ViChaR). As seen in Figure 3.13d, at low injection rates, the FC-CB's performance is almost identical to ViChaR's. However, as network traffic increases, FC-CB's performance starts to degrade compared to ViChaR. This is attributed to FC-CB's fixed number of VCs (i.e., just like DAMQ). Under heavier traffic loads, ViChaR's ability to dynamically dispense more VCs helps improve performance quite drastically. Note also that both FC-CB and DAMQ would incur much higher area and power penalties (as explained in Section 3.2). In terms of throughput (not shown here), ViChaR's improvement over DAMQ and FC-CB is a more modest 5% (on average). However, ViChaR would perform substantially better in latency-critical applications.

Finally, Figure 3.13e depicts the spatial variation in the number of VCs used in the 8×8 MESH, while Figure 3.13f shows the temporal variation over simulation time. The average number of VCs used varies continuously according to network traffic. Figure 3.13e shows the average number of VCs dispensed at each node of the 8×8 MESH network over the whole simulation time at an injection rate of 0.25. As expected, the nodes situated at the middle of the network exhibit higher congestion; ViChaR successfully self-throttled its resources by granting more VCs in these nodes in order to optimize performance.

Figure 3.13f, as the network fills up with packets, the average number of VCs used over all nodes increases accordingly to handle the traffic. These results validate the effectiveness of our FCFS scheme employed in the Token (VC) Dispenser (Section 3.3.2).

3.5 Chapter Summary

NoC performance is directly related to the routers' buffer size and utilization. In this chapter, the author introduced a centralized buffer architecture, called the Virtual Channel Regulator (ViChaR), which dynamically allocates virtual channels and buffer slots in real-time, depending on traffic conditions. Unlike current implementations, the ViChaR can dispense a variable number of VCs at any given time to maximize network throughput.

Simulation results using a cycle-accurate network simulator indicate *performance improvement of around 25%* under various traffic patterns, as compared to a conventional router with equal buffer size, with a modest 2% power increase. Most importantly, though, ViChaR is shown to achieve performance similar to that of a generic router, while using a 50% smaller buffer. This attribute is a testament to ViChaR's efficient dynamic buffer management scheme, and is a result of utmost significance in the NoC arena. Synthesized designs in 90 nm technology indicate that *decreasing the ViChaR's buffer size by 50% leads to area and power savings of 30% and 34%, respectively, with no degradation in performance*.