ElastiStore: An Elastic Buffer Architecture for Network-on-Chip Routers

I. Seitanidis, A. Psarras, G. Dimitrakopoulos
Electrical and Computer Engineering
Democritus University of Thrace, Xanthi, GR67100, Greece

C. Nicopoulos
Electrical and Computer Engineering
University of Cyprus, 1678 Nicosia, Cyprus

***Abstract*—The design of scalable Network-on-Chip (NoC) architectures calls for new implementations that achieve high-throughput and low-latency operation, without exceeding the stringent area-energy constraints of modern Systems-on-Chip (SoC). The router's buffer architecture is a critical design aspect that affects both network-wide performance and implementation characteristics. In this paper, we extend Elastic Buffer (EB) architectures to support multiple Virtual Channels (VC) and we derive *ElastiStore*, a novel lightweight elastic buffer architecture that minimizes buffering requirements, without sacrificing performance. The integration of the proposed elastic buffering scheme in the NoC router enables the design of new router architectures – both single-cycle and two-stage pipelined – which offer the same performance as baseline VC-based routers, albeit at a significantly lower area/power cost.**

I. Introduction

Network-on-Chip technology is already being adopted in the majority of large SoCs for simplifying system integration at the IP-assembly functional verification level – all the way down to physical integration - by alleviating physical routing congestion and simplifying timing closure [1]. NoCs also improve performance by parallelizing communication, offer quality-of-service (QoS) guarantees, and enable flexible system partitioning. The majority of these NoC features can be satisfied by the use of virtual channels (VCs). A physical channel can be used in a time-multiplexed manner by different VCs, provided that each VC owns a separate buffer space [2]. VC-based architectures enable traffic separation and isolation by assigning different traffic classes to different VCs, and they reduce on-chip physical routing congestion by trading off physical channel width and the number of supported VCs, thus, creating a more layout-flexible SoC architecture [3].

The NoC needs to be both scalable, in terms of network functionality and performance, as well as flexible in terms of physical implementation. This requirement motivates us to unify a VC-based architecture that favors NoC scalability with elastic buffering, which eases physical implementation and promises area and power reduction.

Owing to its elastic operation, which is based on simple ready/valid handshakes, elastic buffering is a primitive and simplified form of NoC buffering that can be easily integrated in a plug-and-play manner at the inputs and the outputs of the routers (or inside them) [4], [5], as well as on the network links to act as a buffered repeater. Elastic buffering assumes only one form of handshake on each network channel that cannot distinguish between different flows thus making its operation serial in nature. This feature prevents the interleaving of packets and the isolation of traffic flows, while it complicates deadlock prevention. Due to this limitation, direct support for VCs is abandoned and replaced by multiple physical networks, or implemented via complex and non-scalable hybrid EB/VC buffering architectures [6], [7], [8], which remove the basic property of the EBs to act as stitching elements that can be placed seamlessly anywhere in the NoC.

In this paper, we generalize the operation and the implementation of elastic buffering to support multiple VCs. The proposed architecture, which we call *ElastiStore*, minimizes the number of buffers per channel (flip-flops, or latches, according to the implementation) close to the absolute minimum of one buffer slot per VC, without sacrificing performance. The scalability of the proposed scheme is demonstrated by the integration of ElastiStore in NoC routers that lead to new architectures – both single-cycle and two-stage pipelined – which offer the *same performance* as baseline VC-based routers, albeit at a *significantly lower area/power cost*.

The proposed ElstiStore design is envisioned as an archetypical primitive for future, extremely low-cost NoC router implementations, where the performance and functionality enhancements provided by VCs cannot be sacrificed. In fact, due to protocol-support restrictions, the use of VCs will be *mandatory* in future Chip Multi-Processors (CMP) employing directory-based cache coherence protocols. These coherence protocols require isolation between the various message classes, in order to avoid protocol-level deadlocks. For example, the MOESI directory-based cache coherence protocol requires at least three *virtual networks* to prevent protocol-level deadlocks. A *virtual network* comprises one VC (or a group of VCs) tasked with the handling of a specific message class of the cache coherence protocol [9].

Despite the increased functionality demands, the area/power budgets of individual NoC routers will continue to dwindle as the number of processing elements (and, hence, the NoC size) keeps increasing. The ElastiStore solution aims to precisely reconcile the conflicting and diverging demands of low cost and high performance/functionality.

The rest of the paper is organized as follows: Section II briefly describes elastic flow control, while Section III introduces the ElastiStore architecture. Sections IV and V describe the implementation of ElastiStore and its integration into NoC routers, respectively. Experimental results are presented in Section VII, and conclusions are drawn in Section VIII.

II. Baseline Elastic Channel and Buffers

A baseline elastic channel carries – in parallel to the data wires – two extra control wires (valid and ready), which are required to implement the elastic protocol, as shown in Figure 1(a). The EBs implement the elastic protocol by

978-3-9815370-2-4/DATE14/©2014 EDAA

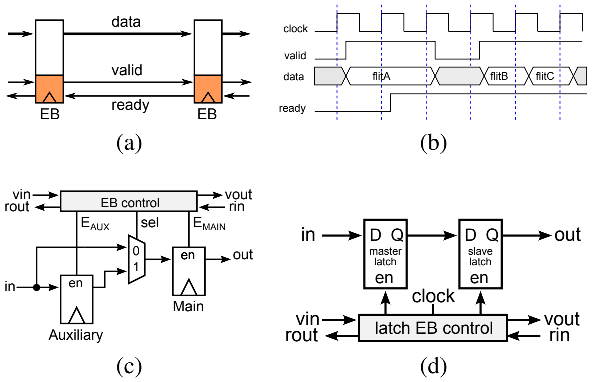


Fig. 1. The fundamentals of the elastic buffering protocol.

replacing any simple data connection with an elastic channel. When an EB can accept an input, it asserts its ready signal upstream; when it has output available, it asserts the valid signal downstream. When two adjacent EBs both see that the valid and ready signals are both true, they independently know the transfer has occurred, without negotiation or acknowledgement. An example of this handshake is shown in Figure 1(b).

When the output of a chain of EBs stalls, the stall can only propagate back one stage per cycle. To handle this, all EBs can hold two words, one for the stalled output, and one caught when necessary from the previous stage. Such an implementation is shown in Figure 1(c). The 2-slot EB can be in three possible states: EMPTY, HALF, and FULL, depending on the number of flits it has stored. By controlling the clock phases accordingly, as shown in [10], the 2-slot EB can be also designed using 2 latches in series, instead of two flip-flops, similar to Figure 1(d). Following the same methodology, any EB architecture derived for edge-triggered flip-flops can be implemented with latches.

III. Elastic VCs

An elastic channel that supports VCs consists of a set of data wires that transfer one flit per clock cycle, and as many pairs of control wires valid(i)/ready(i) as the number of VCs. Figure 2 shows an example of a 2-VC elastic channel. At the protocol level this approach resembles the multiple threads of OCP-IP [11]. Since multiple VCs may be active at the sender, arbitration is employed to select which VC will use the channel. As a result, only one valid(i) signal is asserted per cycle. At the same time, the receiver may be ready to accept flits that can potentially belong to any VC. Therefore, there is no limitation on how many ready(j) signals can be asserted per cycle. The arbiter at the sender should grant only a VC that is ready at the receiver. Therefore, the requests of the active VCs are first qualified by the incoming ready signals.

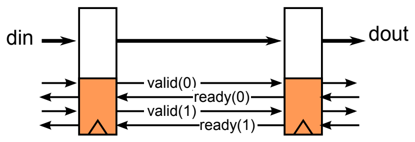


Fig. 2. An example of a 2-VC elastic channel.

A baseline ElastiStore primitive can be built by replicating one EB per VC, including an arbiter and a multiplexer, and following the connections shown in Figure 3 for the case of 3 VCs. The arbiter selects which VC will drive the output by checking if it has valid data and if the corresponding VC is ready downstream. Also, the optional VC qualifiers (1 bit per VC) can enable or disable the request of an active VC. These qualifiers are needed when an ElastiStore is integrated at the inputs of NoC routers.

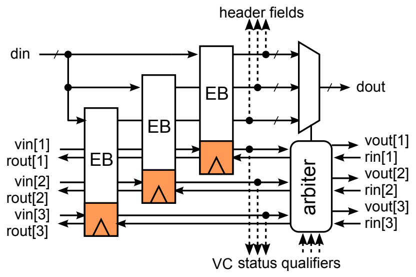


Fig. 3. The baseline ElastiStore architecture.

A 2-slot EB per VC allows each VC to host 2 flits. This is an expensive solution, since the available resources (EBs, in this case) are replicated per VC. When a single VC is active between two baseline ElastiStores, it can achieve 100% throughput. When it stalls, it utilizes the 2 buffer slots available, while the remaining $V-1$ VCs are left un-utilized. When M VCs are active per channel, with $2 \leq M \leq V$, each VC will receive a throughput of $1/M$. In this case of uniform utilization, each VC will use only one buffer out of the two available per VC, since it will be accessed once every M cycles. The second buffer is used only when a VC stalls.

Two buffers per VC are needed in order to allow a single active VC to enjoy full throughput, even if the rest $V-1$ VCs are blocked, occupying their $2(V-1)$ buffers, and assuming that each one of the blocked VCs reserves the right to restart at peak rate. In this paper, we relax this hard constraint and build an ElastiStore using only $V+1$ buffers. Each VC owns a single buffer (V in total), which is enough in the case of uniform utilization, where each VC receives a throughput of $1/M$, with $2 \leq M \leq V$. Furthermore, when a single VC uses the channel without any other VC being active or blocked, i.e., $M=1$, it receives full throughput, and, in the case of a stall, it may use the additional buffer available in ElastiStore. This additional buffer is shared dynamically by all VCs, although only one can have it in each clock cycle.

Figure 4 depicts an example of flit flow on an elastic channel that supports 2 VCs. Initially, all VCs in each stage have one flit available. In the first cycles, each VC receives 1/2 of the throughput per channel ($M=2$), and, at each step, they utilize only one buffer slot. In those cycles, the shared auxiliary registers are not utilized. The shared buffers are used between cycles 4 and 7 to accommodate the stalled words of VC B. In those cycles, VC A – which is not blocked – continues to deliver its words to the output of the channel.

However, when all VCs, except one, are blocked, and the shared buffer is utilized by a blocked VC, then the only active VC will get 50% of the throughput, since it effectively sees only one buffer available per channel. The baseline and “expensive” ElastiStore that allocates 2 buffers to each VC would allow this active VC to enjoy full channel utilization.

ElastiStore offers a reasonable tradeoff, since it saves $V-1$ buffer slots per elastic VC buffer, and limits throughput only under heavy congestion that blocks all the VCs except one.

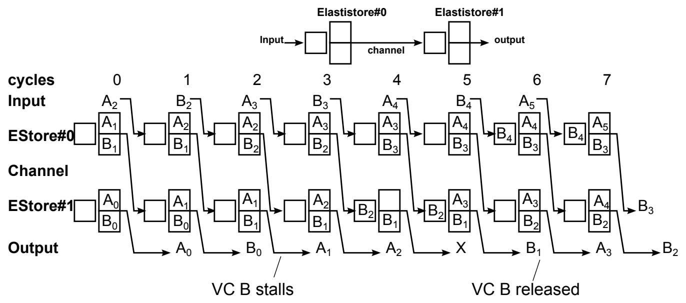


Fig. 4. An example of flit flow on an elastic channel that supports 2 VCs.

In the case of light traffic, a single active VC receives full throughput without any limitation. Also, the static allocation of a single buffer to each VC *guarantees forward progress for all VCs and avoids possible protocol-level deadlocks*.

IV. Implementation of ElastiStore

ElastiStore can be designed using the datapath shown in Figure 5, which consists of a single register per VC along with a shared register that is dynamically shared by all VCs. The select signals of the bypass multiplexers, the load enable signals of the registers, as well as the internal ready/valid signals that are connected to the arbiter and the input interface are produced via ElastiStore control.

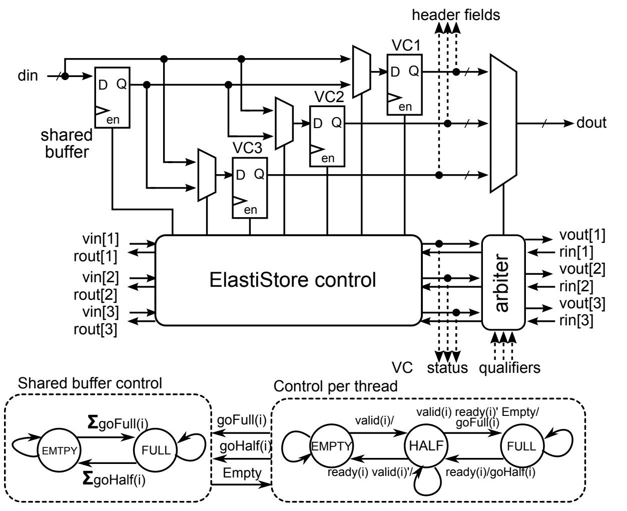


Fig. 5. The datapath and control logic of the proposed ElastiStore architecture. The FSMs of the control logic are shown at the bottom of the figure.

ElastiStore control copies V times the control logic of a single EB, which implements the 3-state FSM shown in Figure 5, allowing each VC to be in the EMPTY, HALF, or FULL states. ElastiStore control tracks the state of each EB, by inspecting the additional goFull and goHalf signals, and guarantees, via the Empty output signal, that only one of them will move to the FULL state. This is needed, since only one VC is allowed to store two flits – in the case of a downstream stall – by using the shared buffer. A two-state FSM associated with the shared buffer tracks this condition and produces the Empty signal, which allows the transition of only one EB control from the HALF to the FULL state.

When a new word arrives at the input and belongs to the ith VC that is in EMPTY state, it is stored in the main register of the ith VC and moves to the HALF state. On the contrary, the VCs in the HALF state are ready to accept new data, as long as no thread is in the FULL state. If this is the case, and new data arrives, three operations take place in the same cycle: (a) the new data word is stored in the shared buffer, (b) its state moves to FULL, and (c) all threads that were in the HALF state stop being ready to accept new data.

When the arbiter selects a VC that is in the HALF state, its data is dequeued from the VC's main register and returns to the EMPTY state. On the contrary, if the selected VC was the only one in the FULL state – having stored two words in ElastiStore (in the main register and the shared buffer) – it should move to the HALF state, after reading the data from the main register. During this state transition, the main register of the VC should be refilled by the data stored in the shared buffer. The shared buffer cannot receive a new word in the same cycle, since its availability – which releases all VCs being in the HALF state – will appear on the upstream channel in the next clock cycle.

Even if the datapath of ElastiStore is reduced to only $V+1$ registers, its control is almost the same as using a 2-slot EB per VC, as shown in Figure 3. Using this property we can design a latch-based ElastiStore by just changing the FSM per EB to its latch-based equivalent [10], and qualifying the enable signals by the appropriate clock phase.

ElastiStore accepts at most one flit per cycle for a ready VC. The registers of the rest VCs can be clock-gated to save dynamic power during their idle cycles. The main register of a VC should be clocked (a) when it moves from the EMPTY to the HALF state (it is accepting a new incoming flit), and (b) when it moves from FULL to HALF, (the main register is refilled by the contents of the shared buffer). On the contrary, the shared buffer and the ElastiStore control logic that keeps track of the state of each VC are always clocked. This form of fine-grained clock gating activates for writing only two data registers, independent of the total number of VCs.

A. Lower Number of Handshake Wires

In an elastic channel, the majority of wires are dedicated to data signals. The percentage of handshake wires increases on narrow data channels that support many VCs. Even if there are abundant on-chip wiring resources to support separate handshake signals per VC, the elastic protocol can still be modified to minimize the number of control wires.

The first modification encodes the V valid signals, from which only one will be active, to a single valid bit per channel and a VCid that encodes the index of the selected VC in $\log_2 V$ bits. The second modification minimizes the number of ready signals. When the ith VC at the receiver is ready to accept a new flit, it asserts the ready(i) signal independently of the rest of the VCs. Therefore, multiple ready signals can be asserted per clock cycle. If we do not want to spend more wires for back-pressure than we do for forward validity notification – i.e., $1+\log_2 V$ wires – we need to store the condition of each downstream VC at the sender. In this "stored-ready" approach, which is effectively a primitive form of a credit-based flow control, the sender keeps track of the readiness of each downstream VC, by checking the value of the local stored ready bits. Thus, the receiver only sends a status update signal and a VCid backwards, which indexes which VC should update its status to ready.

The sender stores V ready bits, one for each downstream VC, and one bit for the shared buffer. In this case, an active

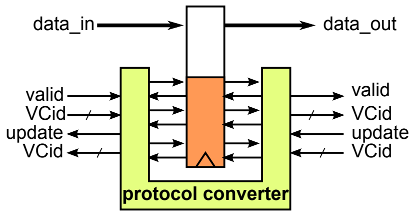


Fig. 6. The implementation of the stored-ready approach is done outside ElastiStore by protocol converters.

VC is eligible to send a new flit when either the local ready bit, or the ready bit of the shared buffer, is asserted. For the VC that was granted, we de-assert its local ready bit. If it was already zero, we de-assert the ready bit of the shared buffer. When an update is received for the ith VC, the ith local ready bit should be asserted. If it was already asserted, this update refers to the ready bit of the shared buffer (i.e., the ith VC has sent two flits in the past, due to a downstream stall, and one of them occupied the shared buffer). The implementation of the stored-ready approach can be done outside ElastiStore by protocol converters, as shown in Fig. 6.

V. Integration of ElastiStore in NoC Routers

When a packet arrives in a router, it needs to find its output destination port via routing computation (RC). The output port can be pre-computed in the previous router, using lookahead RC (LRC). Each packet then has to choose a VC at the input of the next router, before leaving the current router (known as an "output VC"). Matching input VCs to output VCs is performed by the VC allocator (VA). Allowing packets to change VC in-flight can be employed when the routing algorithm does not impose any VC restrictions (e.g., XY routing does not even require the presence of VCs). However, if the routing algorithm and/or the upper-layer protocol (e.g., cache coherence) place specific restrictions on the use of VCs, then arbitrary in-flight VC changes are prohibited, because they may lead to deadlocks. In the presence of VC restrictions, the VC allocator will enforce all rules during VC allocation to ensure deadlock freedom. Such VC restrictions are orthogonal to the operation of ElastiStore.

The flits that own an output VC arbitrate for accessing their output port. If a flit wins this stage – called switch allocation (SA) and organized in local and global arbitration steps, SA1 and SA2 – it will traverse the crossbar (ST – switch traversal), and, then it will move to the output link (LT – link traversal) towards the next router [12].

ElastiStore can be integrated at the inputs, at the outputs, or inside a router as shown in Fig. 7. ElastiStore modules are placed between router stages, thus replacing the conventional pipeline registers. A single-stage NoC router would require at most two ElastiStore modules; one at the input and one at the output. Similarly, a two-stage NoC router would require one additional intermediate ElastiStore module (a total of three) between the first and second stages of the router.

The input ElastiStore utilizes all the optional output signals shown in Fig. 5: The header information from each VC is passed to the RC unit, which computes the output destination. Then, using the VC status information of the ElastiStore, the head flits from each VC try to acquire an output VC in their output destination. The output VCs correspond either to the

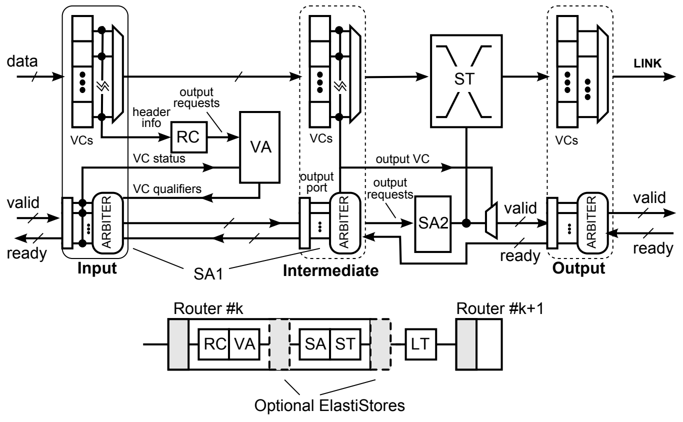


Fig. 7. ElastiStore can be integrated at the inputs, at the outputs, or inside a router.

VC of the output ElastiStore – if it exists – or to the VCs of the input ElastiStore of the next router.

The VCs that have not yet received an output VC cannot leave the input. On the contrary, the input VCs that have been matched to an output VC can move forward and arbitrate for an output port. SA can be performed in the same cycle, or in the next cycle, depending on the presence of the intermediate ElastiStore. SA1 is performed by the arbiter inside the ElastiStore (either at the inputs or in the middle) and SA2 follows to resolve contention for the same output among different inputs.

The intermediate ElastiStore can hold the flits that have acquired an output VC and move forward, waiting to be selected by SA. The flits need to be enhanced with two tags: the output destination port and the output VC acquired via the head flit of the packet. Once inside the intermediate ElastiStore, each flit can independently arbitrate for an output port. The intermediate ElastiStore is just an extension of the input ElastiStore. The flit that leaves the input ElastiStore and moves to the intermediate one stays on the same VC as in the input. A flit moves to its destination output VC only when it leaves the router, or when it is stored in the output ElastiStore by asserting the appropriate valid signal.

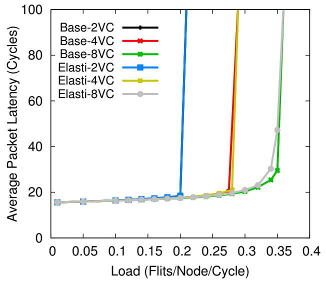
The ready/valid handshake connections are transferred across routers, following the connections of the elastic channels. If a router is equipped with an intermediate or an output ElastiStore, the handshake signals propagate in a step-by-step manner after being registered in each ElastiStore. When ElastiStores are placed only at the inputs, then the intermediate and the output ElastiStores act as feedthroughs for the data and the ready/valid handshakes.

VI. Evaluation

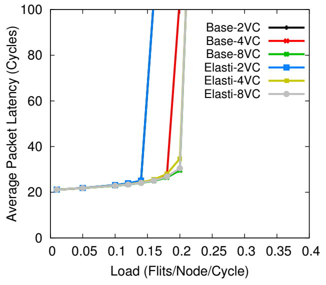
In this section, we compare ElastiStore-based routers with conventional VC-based routers, both in terms of network performance and hardware complexity.

A. Network Performance

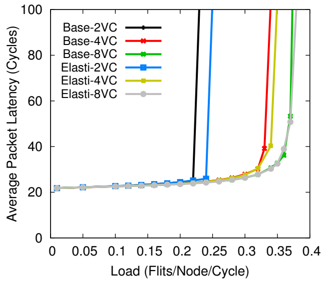
All comparisons have been performed using a cycle-accurate SystemC network simulator that models all micro-architectural components of a NoC router, assuming an 8×8 2D mesh network with XY dimension-ordered routing. Evaluation involves synthetic traffic patterns using uniform random and bit-complement traffic. Other permutation traffic patterns follow very similar trends to bit-complement traffic. The



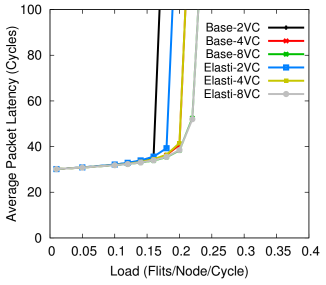
(a) Uniform random traffic



(b) Bit-complement traffic



(c) Uniform random traffic



(d) Bit-complement traffic

Fig. 8. Latency vs. load curves for (a)-(b) single-stage and (c)-(d) two-stage routers.

injected traffic consists of two types of packets to mimic realistic system scenarios: 1-flit short packets (just like request packets in a CMP), and longer 5-flit packets (just like response packets carrying a cache line). For the latency-throughput analysis, we assume a bimodal distribution of packets with 50% of the packets being short, 1-flit packets, and the rest being long, 5-flit packets. This percentage is in accordance with recent studies of cache traffic in CMPs running real application workloads [13].

Baseline VC routers can be built with shallow, or deep buffers per VC. It is critical, however, for each VC to contain as many buffers as needed to cover the credit round-trip latency. In single-cycle baseline routers, each flit spends one cycle inside the router and one additional cycle on the link. To isolate the link from the rest of the router, a simple output buffer (single pipeline register) is added at the output of the crossbar. In this configuration, each VC needs 3 buffers to cover the round-trip time. This configuration is sufficient to achieve high performance and keep the total buffering per router to reasonable levels when the number of VCs is high. For example, a pipelined router with two stages increments the credit round-trip latency by one – unless direct combinational credit update paths are employed across routers, which limit the benefits of pipelining – but it enjoys a higher clock frequency. Therefore, it needs a minimum of 4 buffers per VC. The latency-equivalent of the baseline single-cycle VC-based router is the router with one ElastiStore at the inputs and the outputs, while the two-cycle pipelined router is equivalent to an elastic router that also contains an intermediate ElastiStore. Each ElastiStore costs $V+1$ buffers. Table I summarizes the buffer requirements in each case.

TABLE I
BUFFER COMPARISON OF BASELINE AND ELASTISTORE-BASED ROUTERS.

Stages	Baseline	ElastiStore
1-stage	$3NV+N$	$2(V+1)N$
2-stage	$4NV+N$	$3(V+1)N$

Even with this lower amount of buffering – which translates directly to area/power savings – the ElastiStore-based routers achieve similar network performance when compared to single and two-cycle VC-based routers. Figures 8(a) and (b) depict the latency-load curves of both single-cycle routers under comparison when varying the number of VCs. In all cases, the performance of the routers is indistinguishable both at low and at high loads. The same conclusion is drawn by the results shown in Figures 8(c) and (d) for the case of two-stage routers. Therefore, *the savings of ElastiStore are offered to the NoC designer for free, without trading off performance*.

Multiple physical EB-based networks of simpler routers [4], with each one mapped to one VC, enjoy higher clock frequencies, due to the removal of the VA stage. However, when compared with ElastiStore-based routers under equal network bisection bandwidth, they suffer in performance, as verified by our experiments that are omitted due to space limitation, because of the high serialization latency imposed by the narrower channels per physical network.

Finally, VC-based routers can be simplified with static destination-based VC allocation that does not allow packets to change VC and keep the one given to them at the source. This feature can be also applied to ElastiStore routers. Although this option favors the complexity of the routers, as done in [14], the static allocation of VC (a) reduces the overall throughput by increasing head-of-line blocking per static VC, and (b) complicates adaptive routing.

B. Hardware Implementation

The proposed single-stage and two-stage ElastiStore routers, using lookahead RC, have been implemented in VHDL and mapped to an industrial low-power 40 nm standard-cell library under worst-case conditions (0.8V, 125°C), using the Cadence digital implementation flow. The generic router models have been configured to 5 input-output ports, as needed by a 2D mesh network, and to 4 VCs per port, while the flit width was set to 64 bits. The area/delay curves obtained for all designs – after constraining the logic-synthesis and back-end tools, and the extraction of physical layout information (each output is loaded with a wire of 2 mm) – are shown in Figure 9.

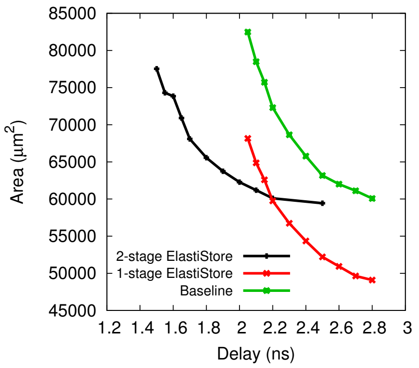


Fig. 9. Hardware implementation results of ElastiStore routers.

Figure 9 includes the performance of a single-cycle 4-VC router with LRC and 3 buffers per input VC and two ElastiStore-based routers with the same number of VCs. Both

ElastiStore routers lie below the baseline router. The single-stage router has the same delay as the baseline VC-based router, albeit with significantly lower area, due to buffer reduction. Both single-stage routers could have improved their speed by using VA-SA speculation [15]. Even though the two-stage ElastiStore router requires more area than the single-stage router at its minimum delay point, it ends up being more area-efficient when the delay constraint is relaxed. At high delays, however, the most area-efficient implementation is the single cycle ElastiStore router.

The measured power consumption follows the trend of the area measurements, assuming that both the baseline and the ElastiStore architectures can clock-gate the unused VCs. Clock gating, in our case, is simpler and more fine grained. Also, since a latch-based implementation is not a unique characteristic of elastic buffering and can also be applied to the design of larger FIFOs, both the baseline and ElastiStore designs can use them to decrease their area/power requirements proportionally.

VII. Related work

ElastiStore is related to two thrusts of prior research: (a) elastic flow control, and (b) shared-buffering in NoC routers.

Elastic buffers were employed in [4] as a low-cost buffering mechanism, which replaces the monolithic buffers of wormhole routers. The support for VCs was offered via multiple physical networks, or via a hybrid EB/VC flow control that employs a combination of EBs (in the links) and regular VC buffers [16], [6]. Similar hybrid techniques have been employed in [8], [7], with the extra feature that the VC buffer space accompanying EBs was shared among VCs. While these solutions successfully allow for separation of traffic flows (facilitated by the VC buffers), they still rely on a single valid/ready interface between the EBs in each link. This interface cannot distinguish the different flows. So, the challenge is to ensure that all the flits of a blocked VC are drained from the EBs in an orderly fashion (i.e., placed contiguously in a VC buffer with no interleaving), so that the EBs in the links may be used by a non-blocked VC. To achieve this goal, the hybrid EB/VC architectures rely on non-trivial control logic that coordinates the VC traffic flows.

Instead, ElastiStore avoids this complication by employing individual handshaking interfaces (or stored readies) for each supported VC, so that the various VC traffic flows are inherently logically separated and easily guided to their respective parallel buffer slots. The presence of a shared buffer in ElastiStore is instrumental in optimizing the use of available buffer space within the NoC router. In essence, the proposed mechanism achieves the same objective as the significantly more expensive shared-buffering and buffer-stealing schemes [17], [18], [19], [20].

In shared-buffer schemes, designers predominantly use either linked lists [18], or table-based approaches [17] to coordinate traffic flow through the buffers. Each VC must maintain its own set of pointers to identify where its flits are located in the buffer, regardless of the size of the buffer, while taking care deadlock avoidance. While the cost of control logic is amortized in routers with large buffer space, the overhead becomes significant in low-cost routers with minimal buffer space. ElastiStore requires no pointer logic whatsoever and avoids protocol-level deadlocks by construction, thereby offering a zero-cost solution to this critical issue.

VIII. Conclusions

As multi-core systems transition to the many-core realm, the pressure on the interconnection network is substantially elevated. The NoC is expected to undertake the expanding demands of the ever-increasing numbers of processing elements, while – at the same time – its area/power footprint remains severely constrained. Hence, low-cost NoC designs that achieve high-throughput and low-latency operation are imperative for future scalability. The NoC router's buffers are major consumers of area and power, and key enablers of high performance. Moreover, buffers are used to facilitate VCs, which are instrumental in further enhancing performance and allowing deadlock freedom. In this paper, we proposed ElastiStore that extends the notion of elastic buffering to multiple VCs. ElastiStore reduces the buffer requirements to nearly the minimum possible, while still achieving the same performance as the much more expensive, traditional VC-based routers.

References

[1] J. Handy, "NoC interconnect improves SoC economics," *Objective analysis - Semiconductor market research*, 2011.

[2] W. J. Dally, "Virtual-Channel Flow Control," in *Proc. of the Intl. Symp. on Computer Architecture*, May 1990, pp. 60–68.

[3] J. Browne, "On-Chip Communications Network Report, 2012."

[4] G. Michelogiannakis, J. Balfour, and W. J. Dally, "Elastic buffer flow control for on-chip networks," in *IEEE Int. Symp. on High Performance Computer Architecture*, 2009.

[5] A. Roca, C. Hernndez, J. Flich, F. Silla, and J. Duato, "Silicon-aware distributed switch architecture for on-chip networks," *Journal of Systems Architecture*, vol. 59, no. 7, pp. 505 – 515, 2013.

[6] G. Michelogiannakis and W. Dally, "Elastic buffer flow control for on-chip networks," *IEEE Trans. on Computers*, vol. 62, no. 2, Feb. 2013.

[7] S. M. Hassan and S. Yalamanchili, "Centralized buffer router: A low latency, low power router for high radix nocs," in *IEEE/ACM International Symposium on Network on Chip*, April 2013.

[8] A. K. Kodi, A. Sarathy, and A. Louri, "ideal: Inter-router dual-function energy and area-efficient links for network-on-chip (noc) architectures," in *Proc. of Intl Symp. on Comp. Architecture*, 2008, pp. 241–250.

[9] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Computer Architecture News*, vol. 33, 2005.

[10] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of Synchronous Elastic Architectures," in *Proc. ACM/IEEE Design Automation Conference*, Jul. 2006, pp. 657–662.

[11] OCP-IP protocol specification. www.ocp-ip.org

[12] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[13] S. Ma, N. Enright Jerger, and Z. Wang, "Whole Packet Forwarding: Efficient Design of Fully Adaptive Routing Algorithms for Networks-on-Chip," in *Proc. of the Intern. Symp. on High Performance Computer Architecture*, Feb. 2012, pp. 467–478.

[14] F. Gilabert and et al., "Improved utilization of noc channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip," in *NOCS*, 2010, pp. 165–172.

[15] R. D. Mullins, A. F. West, and S. W. Moore, "Low-latency virtual-channel routers for on-chip networks," in *Proc. of the Intl. Symp. on Computer Architecture*, 2004, pp. 188–197.

[16] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "A QoS-Enabled On-Die Interconnect Fabric for Kilo-Node Chips," *IEEE Micro*, vol. 32, no. 3, May 2012.

[17] C. Nicopoulos and et al., "Vichar: A dynamic virtual channel regulator for network-on-chip routers," in *IEEE/ACM Intern. Symp. on Microarchitecture*, 2006, pp. 333–346.

[18] M. Lai, Z. Wang, L. Gao, H. Lu, and K. Dai, "A Dynamically-Allocated Virtual Channel Architecture with Congestion Awareness for On-Chip Routers," in *Design Automation Conference*, 2008.

[19] W. Su, J. S. Shen, and P. A. Hsiung, "Network-on-Chip Router Design with Buffer-Stealing," in *ASP-Design Automation Conference*, 2011.

[20] A. T. Tran and B. M. Baas, "RoShaQ: High-performance on-chip router with shared queues," in *IEEE Intern. Conf. on Computer Design*, Oct. 2011, pp. 232–238.