CrossOver: Clock Domain Crossing under Virtual-Channel Flow Control

Michalis Paschou*, Anastasios Psarras*, Chrysostomos Nicopoulos† and Giorgos Dimitrakopoulos*
*Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
†Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus

***Abstract*—Technology scaling, process variations, and/or 3D integration make the design of fully synchronous Systems-on-Chip (SoC) a challenging task. Partitioning the SoC into Globally Asynchronous, Locally Synchronous (GALS) islands – aka clock domains – partially alleviates the difficulties in clock distribution. Such partitioning of the SoC is also necessary when supporting Dynamic Voltage and Frequency Scaling (DVFS) across parts of the system to minimize power consumption. The Network-on-Chip (NoC) is an inherently distributed architecture that is physically spread over the entire chip; thus, it should readily support communication across multiple asynchronous clock domains. In this paper, we generalize the fundamental properties of Virtual-Channel (VC) flow control across asynchronous clock domains. A new set of flow control rules is presented, which lead to efficient and deadlock-free communication, while still respecting the properties of traditional (synchronous) VC-based flow control. The derived flow control policy, called *CrossOver*, opens up a new design space, which is quantitatively explored in this paper. The goal of this investigation is to identify the configuration that maximizes throughput with the least cost, in terms of buffering requirements.**

I. INTRODUCTION

The number of components on a single chip is rapidly growing, due to increasing levels of integration and shrinking transistor geometries. Complex Systems-on-Chip (SoC) typically involve a variety of components that should easily be integrated at design-time, and efficiently communicate at run-time. The Network-on-Chip (NoC) plays a key role in satisfying both requirements.

From an application perspective, NoCs are expected to: (a) parallelize communication, (b) possibly provide Quality-of-Service (QoS) guarantees, and (c) enable system partitioning. Resource separation within the NoC is typically facilitated by assigning different message classes to different Virtual Channels (VCs) [1]. This separation is also instrumental for the correct operation of higher-level protocols (e.g., cache coherence), which require isolation between the various message classes to avoid protocol-level deadlocks [2].

Modern SoCs implemented in deeply scaled technologies face slow wires and Process/Voltage/Temperature (PVT) variations. These challenges make the synchronous abstraction increasingly untenable over large chip areas, thereby requiring immense design effort to achieve timing closure. A fully asynchronous approach [3] would eliminate many of the clocking/variation issues, but current design tools and IP libraries rely heavily on the synchronous paradigm. Thus, asynchronous NoCs tend to be very complex to design and validate.

Alternatively, the Globally Asynchronous, Locally Synchronous (GALS) design methodology mitigates the difficulty of global timing closure. The SoC comprises local islands of fully synchronous designs (called *clock domains*), with different blocks of the system operating asynchronously to each other [4, 5]. Such partitioning is also required in Dynamic Voltage and Frequency Scaling (DVFS) architectures. The voltage and frequency of certain portions of the system change (independently) according to the workload, in order to minimize power consumption [6, 7]. Asynchronous clock domains can have arbitrary frequency and phase relationships. Signals that cross these clock-domain boundaries – called *Clock Domain Crossing (CDC)* – have to be synchronized before they can be used in the receiving domain [8].

Synchronization is usually achieved by: (a) synchronizer library cells (i.e., *brute-force synchronizers*, typically used for single-bit signals); (b) req/ack handshaking – in parallel to data buses – which suffers from throughput loss, due to the latency overhead incurred by the handshaking protocol; and (c) by using FIFO synchronizers, which achieve full-throughput data transfers after appropriate FIFO buffer depth selection [8].

A GALS and/or DVFS NoC implementation can take many forms. In one scenario, all logic in the NoC operates synchronously on a dedicated clock, while the connecting IP blocks operate asynchronously in a different clock domain. The decoupling between the IP and NoC clocks is performed at the network interface. However, providing a single clock reference to all NoC components is challenging, since the NoC is a distributed architecture physically spread over the chip. A more flexible approach would enable asynchronous CDC in any part of the NoC. In this case, the NoC itself would be composed of different sectors operating within completely asynchronous clock domains. This is achieved by allowing any point-to-point link between any two routers in the NoC to belong to different clock domains.

CDC using FIFO synchronizers includes some form of flow control to enable lossless data transfers [9, 10]. While the *Xon/Xofff* (aka *stall/go*, or *ready/valid*) scheme is directly applicable to FIFO synchronizers, it is inefficient when applied to CDC with VC buffers. The mismatched asynchronous clocks on either side of the boundary necessitate the use of dynamic thresholds for the initiation and termination of transmission, in order to ensure lossless data transfers. Since the mismatch in clock frequencies can vary, the Xon/Xoff thresholds are different for each combination of sender/receiver clock frequencies. Hence, the VC buffers must be large enough to absorb all in-flight data, covering the worst-case clock difference, which could become prohibitively expensive.

On the other hand, *credit-based* flow control is lossless, by construction, and can operate with arbitrarily small VC buffers, irrespective of the worst-case frequency mismatch across the clock boundary [1]. The buffer slots (per VC) and the clock frequency mismatch only determine the throughput of data transfers. Therefore, credit-based flow control is the best-suited policy for implementing asynchronous CDC across VCs. Despite the large volume of work in synchronization and GALS for NoCs [5, 11, 12], existing work does *not* include VC flow control, or credits. A few industrial papers [13–15] have dealt with asynchronous CDC under credit-based VC flow control, but they provide very little detail pertaining to their implementation. On the contrary, VC flow control across *mesochronous* interfaces, where clock frequency and phase relationships are known [16], is more clearly examined [17, 18].

In this paper, our goal is to elucidate the intricacies of CDC in VC-based NoCs employing credit-based flow control. The proposed approach, called ***CrossOver***, leverages existing asynchronous FIFO synchronizers to provide deadlock-free communication across asynchronous clock domains, by tackling the protocol-level implications of VC flow control in an asynchronous CDC environment. In the case of bidirectional interfaces, CrossOver enables consolidated structures that can jointly synchronize data and credits transferred across routers. This approach enables a single point of synchronization in each clock domain for all signals, thereby increasing reliability and simplifying the verification of CDC.

To the best of our knowledge, this work is **the first to analyze the implications of asynchronous CDC on the flow control policy of VC-based NoCs**. The CrossOver flow control policy and its hardware implementation open up a new design space that involves various micro-architectural features and buffer-sizing choices. The quantitative analysis presented in this paper reveals the interplay between the micro-architectural knobs and their effects on the area and throughput of the design. More importantly, CrossOver's novel micro-architectural features – including a new *credit accumulation* scheme – yield very cost-efficient hardware implementations.

II. VC Flow-Control Synchronization Interface

To divide a physical channel into V VCs, the input queue at the receiver must be separated into as many independent queues as the number of VCs. To implement VC flow control using credits, the sender keeps a credit counter for each downstream VC. A new flit that belongs to the ith VC can be sent on the channel, as long as $creditCount[i] > 0$, i.e., there is at least one empty slot in the downstream buffer for the ith VC. Since the state of each VC is kept at the sender, the receiver only needs to send backwards a credit-update signal, including a VC ID, which indexes the VC that has one more available credit for the next cycle. On a credit update referring to the jth VC, the corresponding credit counter is increased.

With credits, lossless operation is guaranteed, irrespective of VC-buffer depth, since no flit can be in-flight if it has not consumed a credit beforehand. The number of buffer slots per VC merely determines the throughput of data transmission.

When the sender and the receiver of a VC flow-controlled link belong to the same clock domain, all data transfers and credit updates occur synchronously on the positive (or negative) edge of the clock. In the case that the sender and the receiver belong to different clock domains, as shown in Figure 1, the forward data signals launched by the sender should be synchronized to the receiver's side, while the credit-update signals that are transferred in the opposite direction should be synchronized to the sender's clock domain.

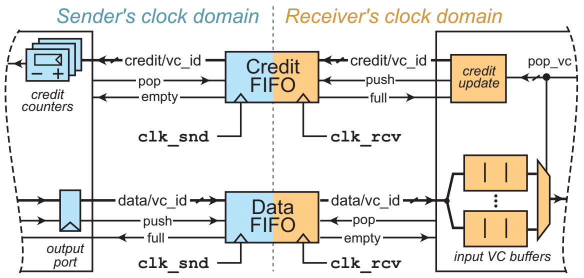

Fig. 1. The organization of an asynchronous VC flow-controlled link, where the sender and the receiver belong to different, unrelated clock domains. The signals exchanged (data and credit updates) are synchronized to the receiving domain using a dual-clock FIFO. FIFO synchronizers are always read (popped), as long as they are not empty.

To achieve this goal, two FIFO synchronizers (i.e., dual-clock, or bisynchronous FIFOs) are added, one in each direction, as depicted in Figure 1. A new flit, together with its VC ID, is written into the data sync FIFO first (shared across VCs), and then – after a finite amount of time – it moves to the corresponding VC buffer downstream. In the opposite direction, a credit returned by the receiver is written into the credit sync FIFO and extracted by the sender later on.

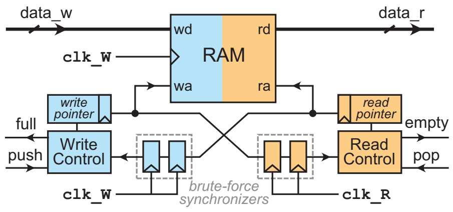

Fig. 2. The internal organization of the dual-clock FIFOs.

Both FIFO synchronizers of Figure 1 follow the organization depicted in Figure 2. Lower latency FIFO synchronizers could also be utilized [10]. New incoming data is written using the sender's clock (write clock) into the position indexed by the write pointer. New data is read out of the FIFO in the receiver's clock domain (read clock) by extracting the data stored in the address indexed by the read pointer. The determination of the state of the FIFO (full, or empty) requires the synchronization and comparison of the read and write pointers. Synchronization of the read and write pointers is performed using brute-force synchronizers, provided that the pointers are gray-coded, so that any synchronization error does not disrupt the pointer location by more than one increment. Brute-force synchronizers do not eliminate the possibility of metastability; they can only reduce it to a negligible value.

Dual-clock FIFOs introduce extra latency compared to a synchronous design, due to the latency incurred by the brute-force synchronization of the gray-coded read and write pointers. Two cycles are spent for pointer synchronization (assuming a 2-stage brute-force synchronizer), and 1 cycle for the read/write operations in each domain. Increasing the depth of the brute-force synchronizers to reduce failure rates would incur a higher latency. Overall, the latency includes 3 cycles of the write clock domain, and 3 cycles of the read clock domain. If the dynamically-changing clock frequencies between the two clock domains can become equal (i.e., each read cycle costs, in time, as much as a write cycle), then a 6-slot deep

FIFO should be used to provide 100% throughput; 6 slots are enough, since they fully cover the 6-cycle (3+3) latency of each domain. However, if the ratio between the two clock frequencies cannot reach the worst case of one (i.e., one clock domain is always faster), then shallower FIFO synchronizers are enough to sustain full throughput.

One alternative to the organization shown in Figure 1 is to use multiple parallel FIFO synchronizers (each VC buffer is an independent dual-clock FIFO). However, there are three compelling reasons why such an approach is sub-optimal: (1) implementing multiple CDC points across a single clock boundary is an ill-advised design practice, which significantly increases the verification effort; (2) such static VC segregation prohibits the use of VC buffer sharing, which is a highly desirable feature (it optimizes and maximizes buffer utilization) in modern NoC designs [19]; (3) the hardware cost is quite high, since, as already shown, each FIFO synchronizer must be deep enough (6 slots at least) to account for the worst-case clock frequency ratio. Therefore, using a common FIFO synchronizer for all VCs before the input VC buffers is the preferable choice. The area savings achieved by this approach and the interplay between VC buffer depth and throughput are presented in Section IV.

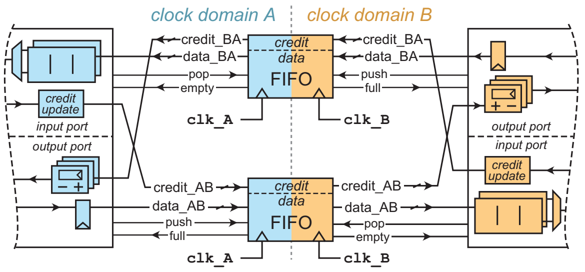

Fig. 3. Bidirectional consolidated CDC interfaces, which merge the synchronization of credit and data in opposite directions.

The majority of NoC topologies include bidirectional links across connected routers. Bidirectionality is typically facilitated using a pair of uni-directional links. In this case, illustrated in Figure 3, both routers can exchange data and credits in opposite directions. For example, router A receives the *data* sent by router B (and written into A's VC buffer), *and* the *credit* updates sent by router B (referring to B's VC buffers). The opposite occurs in the other direction. For maximum design safety and ease of verification, the data and credits that flow from router A to B (and vice versa) should be consolidated and synchronized at one entry point at the inputs of B (and A, respectively), as a merged entity. In each direction, a consolidated FIFO synchronizer carries both the data and credits that need to be synchronized across the two clock domains. Each entry of the consolidated FIFO synchronizer can carry a new flit, or a returning credit, or both, in the same cycle. Consequently, the consolidated credit and data FIFO synchronizers have the same size.

III. The CrossOver Flow Control Policy

VC flow control between two asynchronous clock domains requires a new set of rules, both in the forward and the backward directions. Said rules would allow for deadlock-free communication across the clock boundary.

A. Flow-Control Rules of the Forward Data Path

The ith VC at the sender side is eligible to send a new flit, as long as: (a) $creditCount[i] > 0$, and (b) the data FIFO synchronizer is not full. When both conditions are satisfied, the flit is released from the sender, it is written to the data FIFO synchronizer, and the ith credit counter is decremented.

Since the credits consumed by each flit refer to the VC buffers of the receiver, and not to the buffer slots of the data FIFO synchronizer, it is certain that a flit remains in the data FIFO synchroniser temporarily. At some point in time, the receiver will read out the contents of the data FIFO synchronizer and place the corresponding flit into the appropriate VC buffer.

Even if the flits of potentially all VCs are serialized inside the data FIFO synchronizer, it is impossible for an older flit to perpetually block a newer flit of the same – or of a different – VC, since every flit written to the data FIFO synchronizer will eventually be read out (as previously explained). How fast this sinking happens would depend on the clock frequencies of the two connected domains. Therefore, the data FIFO synchronizer merely acts as a delay element to the channel, without creating any true dependencies across VCs, even if it can become full at some point in time and block the transfer of flits. This condition holds, as long as credit updates return to the sender without any complication. The implications of this requirement will be discussed in Section III-B.

It is important to note that, even though the data FIFO synchronizer is structurally shared across VCs, it *must not* be considered as a shared buffer for the receiver's VC buffers, i.e., none of the available credits should refer to the buffer slots of the data FIFO synchronizer. If such functionality is required (i.e., the FIFO synchronizer being part of the regular VC buffer space of the receiver), the FIFO synchronizer should then be able to read out a flit, even if it is behind the head-of-line position. This would be mandatory, in order to avoid introducing deadlock-inducing dependencies among VCs.

To understand this problem, let us consider a simple example, where 2 VCs on the receiver side – each one with 3 buffer slots – receive data through a 3-slot FIFO synchronizer. The FIFO synchronizer's 3 slots are considered as extra shared buffering for the two VCs. Let us also assume that VC#0, which cannot currently dequeue a flit at the receiver, has used all 3 of its private buffer slots, and 2 of the 3 slots of the data FIFO synchronizer (separate credits are dedicated to these shared buffer slots). At this point, a flit for VC#1 (which is empty) arrives and wishes to move to its dedicated buffer. However, for this to occur, the flit of VC#1 must first be placed at the back of the data FIFO synchronizer (behind the flits of VC#0), and it will be forced to stay there until VC#0 dequeues all of its flits from the FIFO synchronizer. The shared FIFO synchronizer is now full, blocking any other flits from arriving at the receiver, thereby introducing a dependency between VCs #0 and #1. In order to release this dependency, the flit of VC#1 at the back of the FIFO synchronizer should bypass the other two flits of the data FIFO synchronizer and move to the VC#1 buffer. Such necessary bypass capability, however, would **endanger the CDC safety properties** and the validity of the synchronized data. Such situations should certainly be avoided to **safeguard against metastability** problems.

B. Backward Path: Credit Update and Accumulation

In a synchronous VC flow-control implementation, the receiver is obliged to update the sender about the availability of a new credit, as soon as a flit is dequeued from the receiver's input VC buffers. After one cycle (or multiple, when the links are pipelined), the sender is informed about this credit update, and it can immediately reuse it to send a new flit.

On the contrary, when the sender and the receiver are placed in different clock domains and decoupled via FIFO synchronizers, the receiver should include additional functionality so that credits are not lost. The problem arises when the *credit* FIFO synchronizer becomes full, as a result of the sender operating at a lower frequency than the receiver. In such cases, the receiver cannot send more credit updates to the sender (due to the full credit FIFO synchronizer), even if the receiver has generated new credit updates (new flits have been dequeued from the receiver's VC buffers). Thus, proper actions are required, in order to guarantee that credits are not lost and the status of the VC buffers is correctly maintained.

The *pessimistic* solution to the credit-update problem arising in VC flow control across asynchronous clock domains is to select the size of the credit FIFO appropriately, so that it never gets full (irrespective of the sender and receiver clock frequencies). The size of the credit FIFO synchronizer can be bounded, by recognizing the fact that the number of credit updates that can ever be produced cannot be greater than the number of all VC buffer slots. Therefore, in a pessimistic approach, the credit sync FIFO depth *cannot be smaller than the total number of buffer slots at the receiver,* B_{tot}. If each VC at the receiver has a fixed depth (i.e., no shared buffering technique is employed), then the total number of buffer slots at the receiver is $B_{tot} = V \times D$, where V is the number of VCs per port, and D is the buffer depth per VC. Hence, by selecting a credit sync FIFO with depth equal to B_{tot}, we are certain that the credit sync FIFO will never be full and that the receiver can always send the generated credit updates backwards using the same protocol as in the case of synchronous VC flow control. Note that if consolidated FIFO synchronizers use this pessimistic approach, B_{tot} buffer slots would be required to hold both credits and data bits, thus leading to a prohibitively expensive solution.

Alternatively, when the size of the credit FIFO synchronizer is smaller than the total number of buffer slots at the receiver, B_{tot}, the credit FIFO may temporarily become full, thereby prohibiting the receiver from sending the necessary credit updates backwards. In this case, we choose to *accumulate* the generated credits at the receiver, and send them to the sender once the credit FIFO synchronizer has space.

To support ***credit accumulation***, the receiver must include V credit accumulators, each one gathering the unsent credits for each VC. When a flit is dequeued from the ith VC buffer, it causes an increment in the ith credit accumulator. When all credit accumulators are zero and the credit FIFO synchronizer is not full, credit accumulation is bypassed and the credit released is written directly to the credit FIFO synchronizer. On the contrary, when the credit FIFO synchronizer becomes full for a certain period of time, the fast receiver accumulates all the produced credit updates. Once the credit FIFO synchronizer is again not full, the receiver has four choices in sending the accumulated credits backwards:

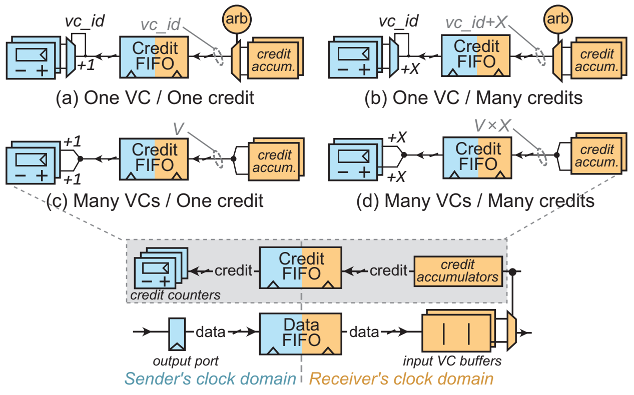

Fig. 4. The organization of the four credit update/accumulation policies.

- ***One VC / One Credit:*** Select one VC, i.e., the ith one, with at least one accumulated credit, and send backwards only one credit update. This action would increment the ith credit counter at the sender and decrement the ith credit accumulator at the receiver.
- ***One VC / Many Credits:*** Select one VC, i.e., the ith one, with at least one accumulated credit, and send all the accumulated credits backwards in the same cycle This action would increase the ith credit counter at the sender by the number of accumulated credits, i.e., X, and decrease to zero the ith credit accumulator at the receiver.
- ***Many VCs / One Credit:*** Send only one credit update from *each and every* VC that has at least one accumulated credit. Since multiple increments are sent backwards in the same cycle, it means that multiple credit counters (accumulators) at the sender (receiver) should be incremented (decremented) in the same cycle.
- ***Many VCs / Many Credits:*** Send backwards all the accumulated credits of all VCs in the same cycle. All credit accumulators are zeroed at once, and multiple credit counters are incremented by their corresponding received credits.

The credit-update paths for the four possible discussed choices are depicted in Figure 4. In the first two choices, arbiter and (de)multiplexing modules are required to select which credit accumulator/counter should be enabled. In the remaining two options, credit update signals involve all VCs simultaneously, thus the multiplexing logic is not involved.

C. Deadlock Freedom

The presence of FIFO synchronizers in both directions of a VC flow-controlled link (i.e., the forward data path, and the backward credit-update path) form a cyclic dependency chain, which is not present in a synchronous implementation. However, a deadlock *cannot* occur, provided that: (a) each flit consumes a credit referring to the downstream VC buffers, before leaving the sender, and (b) the data FIFO synchronizer does not constitute a shared extension of the VC buffers at the receiver (i.e., a position in the data FIFO synchronizer does not refer to any of the available credits).

On any link with B_{tot} credits, B_{tot} items can be present either in the form of real flits (credit consumers), or in the form of real credits (credit producers/updates). These items can be placed in any of the three buffers situated between the sender and the receiver: (1) the normal VC buffers at the receiver (of size B_{tot} buffer slots), (2) the data FIFO synchronizer, or (3) the credit FIFO synchronizer. Provided that the FIFO synchronizers have a non-zero depth, the total number of empty slots is always greater than B_{tot}. Therefore, at least one empty slot is present in the loop of buffers, which prevents the formation of deadlock and prohibits any item (flit, or credit) to be permanently blocked.

IV. Analysis & Experimental Results

In this section, we analyze the operation of CrossOver, and derive the minimum buffering requirements for full throughput. Comparisons to other closely related solutions are also made. Next, we evaluate the behavior of the credit update/accumulation policies using NoC-level simulations.

A. *Buffering and the Asynchronous Round-Trip Time*

Achieving 100% throughput and uninterrupted transmission over a link requires that the receiver owns enough buffer slots to cover the Round-Trip Time (RTT). The RTT is defined as the number of elapsed cycles from the instance the sender puts a flit on the link, until the sender is notified that the receiver has successfully extracted it from its buffer. When the sender and the receiver belong to different clock domains, the RTT – aptly called *asynchronous RTT* (aRTT) – is calculated as the sum of elapsed $\#snd\_cycles + \#rcv\_cycles$. Normalizing aRTT to the *fastest* of the two domains, we get $a\text{RTT} = mC_{SND} + mC_{RCV}$, where $C_{SND} = \frac{f_{SND}}{\max(f_{SND}, f_{RCV})}$ and $C_{RCV} = \frac{f_{RCV}}{\max(f_{SND}, f_{RCV})}$ are the relative speed ratios, and m represents the latency – in cycles – in the forward (data) and backward (credit update) paths. Parameter m depends on the synchronization latency of the dual-clock FIFO, and the latency in data writing and credit-update launching.

As analyzed in Section II, each dual-clock FIFO imposes a 3-cycle latency to both the sender and the receiver domains: 2 cycles are spent for pointer synchronization and 1 cycle for the read/write operations. Outside the dual-clock FIFOs, one cycle is spent in the forward path for writing incoming data into the VC buffers, and one for launching the credit update in the backward path. Overall, $a\text{RTT} = 4C_{SND} + 4C_{RCV}$.

In asynchronous CDC, the clock frequencies of the two domains may change dynamically and can assume any value. The aRTT value is dependent on the ratio of the clock frequencies across the two domains. Figure 5(a) depicts the value of aRTT as a function of the normalized clock-frequency ratio between the sender and the receiver, C_{SND}/C_{RCV}. The aRTT reaches its maximum value when the sender and the receiver employ equal clock frequencies, and diminishes fast when the difference in clock frequencies becomes larger. The maximum aRTT in the proposed configuration is 8.

The minimum buffering required to achieve peak (100%) throughput – regardless of the traffic distribution across the VCs and the clock-frequency ratio across domains – is equal to the maximum value of the aRTT, i.e., at least 8 slots per VC should be provided. This result is verified by the throughput measurements presented in Figure 5(b). This figure depicts the achieved throughput as a function of the clock frequency ratio between the sender and the receiver, for various VC buffer depths. The FIFO synchronizers are 6-slot deep to ensure that they are not limiting throughput in any way. The results in Figure 5(b) assume that all traffic is directed to a single VC at the receiver. Such skewed traffic behavior is

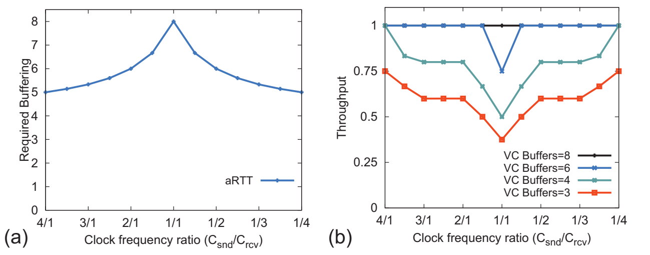

Fig. 5. (a) The aRTT as a function of the clock frequency ratio between the sender and the receiver. (b) The impact on throughput of VC buffer depth, assuming the traffic is skewed toward one VC at the receiver.

more meaningful for this experiment, due to the high buffering demands. Instead, when the traffic is uniformly distributed to all VCs, the buffering requirements (for full throughput) are less demanding. The credit-update policy follows the *Many VCs / One Credit* rule explained in Section III-B.

Note that, in CrossOver, the 8 slots of bufferring per VC are not required to be assigned statically to each VC. Instead, the VC buffer space can be efficiently shared to significantly lower the total buffer area. For example, each VC can be statically assigned with a single buffer slot (to avoid starvation and protocol-level deadlocks), and the remaining slots can be shared among all VCs. As long as each VC has access to at least 8 buffer slots (private+shared) when it has to absorb all the traffic, then full throughput is still guaranteed [19].

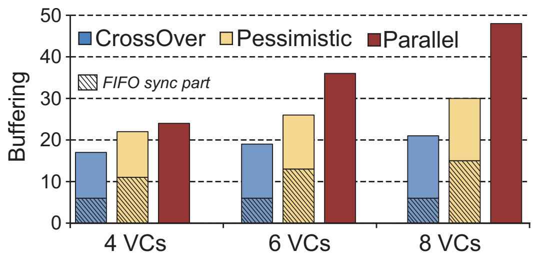

Fig. 6. The worst-case buffering requirements to allow for 100% throughput, when using (a) one dual-clock FIFO per VC ("Parallel"), (b) a single pessimistically-sized FIFO synchronizer ("Pessimistic"), and (c) the proposed "CrossOver" architecture. All cases assume possibly equal clock frequencies in the two domains, i.e., a worst-case aRTT is possible.

Figure 6 illustrates the total number of buffer slots required in the FIFO synchronizers and the VC buffers at the receiver, for three possible implementations: (a) a CDC interface that employs one dual-clock 6-deep FIFO per VC ("Parallel"); (b) a CDC interface with a single pessimistically-sized FIFO syncrhonizer ("Pessimistic"), i.e., a FIFO synchronizer depth equal to the total number of buffer slots at the receiver; and (c) a CrossOver interface that merely covers the aRTT using a single 6-deep FIFO synchronizer. All setups achieve 100% throughput, while the (b) and (c) organizations employ VC buffer sharing [19], which is prohibited in the (a) case. In Figure 6, it is assumed – without loss of generality – that the clock frequencies of the two clock domains can become equal, which dictates the worst-case buffering requirement (since the aRTT value is maximized). In all cases shown in Figure 6, CrossOver requires the least bufferring, with the savings ranging from 22% to 56%. The savings are facilitated by CrossOver's flow control policy, which allows VC buffer sharing (as opposed to "Parallel"), while also ensuring that the size of the FIFO synchronizers is independent of the number of VCs (as opposed to “Pessimistic”).

B. Credit Update/Accumulation Policies

To evaluate the behavior of the credit update/accumulation policies, we employ cycle-accurate simulations on a 4×4 2D-mesh NoC, where each link supports 2 VCs. Networks with more VCs have been tested with indistinguishable results. The network is split in two parts (north-south), with each one operating on a different clock frequency. The CrossOver interfaces are inserted across the clock domain boundaries in the X-axis bisection. Consolidated bidirectional interfaces are used in all cases, while the buffer sizes in all CrossOver interfaces are selected to cover the worst-case aRTT of possibly equal clock frequencies, i.e., 6-deep consolidated FIFO synchronizers and 8 slots/VC for the input VC buffers are used. We present simulation results for synthetic uniform random traffic. Other traffic patterns have been tested and give similar results. Packet lengths follow a bimodal distribution, with half the packets being 1-flit long, and the other half being 5-flit long.

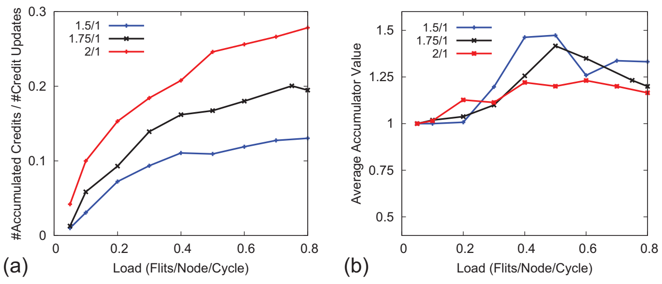

Fig. 7. (a) The average percentage of credits that get accumulated at the clock boundaries, rather than being immediately returned to the sender. (b) The average number of accumulated credits in all CrossOver interfaces.

Our goal is to highlight the behavior of credit accumulation by measuring: (a) the percentage of all credit updates – generated on the NoC links – that are *not* returned (i.e., they get *accumulated*), as a function of input injection load, and for various clock frequency ratios across the clock domains; and (b) the average number of credits being accumulated in the whole network, under the same conditions.

Figure 7(a) depicts the average percentage of credits that are accumulated (and not returned directly to the sender side of the clock boundary), over the total number of generated credit updates, across all CrossOver links. Three clock frequency ratios are measured, while the input injection load corresponds to the load assumed by the slower domain. All four credit return scenarios of Figure 4 have been investigated. All policies exhibit identical behavior in terms of how many credit updates get accumulated overall, without experiencing any other difference in terms of network performance, i.e., latency vs. injection load. As expected, the percentage of credit updates that are actually accumulated increases with the input injection load. The rate of increase diminishes when network saturation is reached. The obtained values are also dependent on the clock frequency difference between the two clock domains. As long as the speed of the fast clock domain increases, the need for credit accumulation becomes larger.

Even if credit accumulation has been shown – in Figure 7(a) – to occur regularly, the average *value* of the credits actually accumulated is, in fact, very low. As depicted in Figure 7(b), the average value of accumulated credits is between 1 and 1.5, irrespective of the clock frequency ratio. Again, all four possible credit-return strategies have been tested, and they exhibit indistinguishable behavior. This result leads us to the conclusion that credit-update strategies aiming to return credits in batches do *not* actually offer any benefits. Instead, the strategies that return one credit to one VC, or to multiple VCs, are much more attuned to the network’s behavior. The main reason for this is that, even if multiple credits can be returned to the sender at once, the slow sender cannot send/transmit to the fast receiver more than one flit per cycle.

V. Conclusions

This paper has identified and analyzed the intricacies involved in the careful and deadlock-free application of VC flow control across asynchronous clock domains. The generalization of credit-based VC flow control requires several changes to the rules of the traditional (synchronous) version, which involve efficient credit accumulation policies and correct usage of the FIFO synchronizers’ buffers. For the first time, to the best of our knowledge, the design space of CDC under VC flow control has been explored analytically and quantitatively, leading to novel efficient hardware implementations for the synchronization interfaces. In the future, we plan to derive cost-efficient plesiochronous and mesochronous VC-based interfaces, by taking advantage of the a priori knowledge of the clock frequency and phase relationships of the clock domains.

References

[1] W. J. Dally, “Virtual-Channel Flow Control,” in *ISCA*, 1990, pp. 60–68.
[2] M. Martin, “Token coherence,” Ph.D. diss., Univ. of. Wisconsin, 2003.
[3] A.Ghiribaldi, D.Bertozzi, and S.Nowick, “A transition-signaling bundled data NoC switch architecture for cost-effective GALS multicore systems,” in *Proc. of DATE*, 2013, pp. 332–337.
[4] W. J. Dally, C. Malachowsky, and S. W. Keckler, “21st century digital design tools,” in *Proc. of DAC*, 2013.
[5] G.Campobello and et al., “GALS networks on chip: a new solution for asynchronous delay-insensitive links,” in *DATE*, 2006, pp. 160–165.
[6] U.Ogras and et al., “Voltage-frequency island partitioning for GALS-based networks-on-chip,” in *DAC*, 2007, pp. 110–115.
[7] E.Beigne and et al., “Dynamic voltage and frequency scaling architecture for units integration within a GALS NoC,” in *NoCS*, 2008, pp. 129–138.
[8] R. Ginosar, “Metastability and synchronizers: A tutorial,” *IEEE Design & Test of Computers*, vol. 28, no. 5, pp. 23–35, 2011.
[9] R.Apperson and et al., “A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains,” *IEEE Trans. on VLSI*, pp. 1125–1134, 2007.
[10] B.Keller, M.Fojtik, and B.Khailany, “A pausible bisynchronous FIFO for GALS systems,” in *Proc. of ASYNC*, May 2015, pp. 1–8.
[11] T.Jain and et al., “Asynchronous bypass channels for multi-synchronous nocs: A router microarchitecture, topology, and routing algorithm,” *IEEE Trans. on CAD*, pp. 1663–1676, 2011.
[12] Z. Lu, “Cross clock-domain tdm virtual circuits for networks on chips,” in *Proc. of Int. Symp. on NoCs*, May 2011, pp. 209–216.
[13] J. Bainbridge, S. Hamilton, and N. Wingen, “Synchronizer with a timing closure enhancement,” Patent US 0 239 842, Dec., 2013.
[14] J. Philip, J. Rowlands, and S. Kumar, “Multiple clock domains in NoC,” Patent US 0 376 569, Dec., 2014.
[15] J. Howard and et al., “A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS3,” in *ISSCC*, 2010, pp. 58–59.
[16] D. Verbitsky and et al., “Starsync: An extendable standard-cell mesochronous synchronizer,” *Integration*, no. 2, pp. 250–260, 2014.
[17] S. Saponara and et al., “LIME: A low-latency and low-complexity on-chip mesochronous link with integrated flow control,” in *Euromicro DSD*, 2008, pp. 32–35.
[18] Y. Hoskote and et al., “A 5-GHz mesh interconnect for a teraflops processor,” *IEEE Micro*, pp. 51–61, 2007.
[19] C. Nicopoulos and et al., “Vichar: A dynamic virtual channel regulator for network-on-chip routers,” in *Proc. of MICRO*, 2006, pp. 333–346.