Virtual-Channel Flow Control Across Mesochronous Clock Domains

Giorgos Dimitrakopoulos, Anastasios Psarras
Electrical and Computer Engineering
Democritus University of Thrace, Xanthi, Greece

Chrysostomos Nicopoulos
Electrical and Computer Engineering
University of Cyprus, Nicosia, Cyprus

***Abstract*—The Network-on-Chip (NoC) is physically spread over the entire chip and it should readily support communication across multiple clock domains. In NoCs, the synchronization required for sending data across clock domains should be seamlessly combined with the flow control rules that govern data transfer on NoC links. In this work, we focus on implementing virtual-channel flow control on links that connect NoC components placed across mesochronous clock domains. In mesochronous interfaces, the connected modules receive the same clock signal but the edges of the arriving clock signals exhibit an unknown phase relationship. To this end, we propose FastCross a loosely-coupled synchronization architecture for NoC links with VC flow control that simplifies hardware implementation by enabling VC buffer sharing and interface consolidation.**

***Index Terms*—Network-on-Chip, Virtual-channel flow control, Clock-domain crossing, Mesochronous interfaces.**

I. Introduction

Technology scaling comes with significant challenges due to slow wires and Process/Voltage/Temperature (PVT) variations. These challenges make fully synchronous design increasingly untenable over large chip areas. Being innately distributed across the chip, the Network-on-Chip (NoC) – the on-chip communication backbone of modern SoCs – is also afflicted by these issues. In this context, the Globally Asynchronous, Locally Synchronous (GALS) design methodology mitigates the difficulty of global timing closure [1]. In this case, the chip is composed of islands (clock domains) that operate under a fully synchronous approach, while the communication between domains is done asynchronously [2], [3], [4].

In this work, we focus on communication across mesochronous clock domains, where the clocks of each domain are driven by partially independent clock trees that operate under the same frequency, but with an arbitrary but fixed phase difference [5]. Signals that cross these clock-domain boundaries – a process commonly referred to as *Clock Domain Crossing (CDC)* – have to be synchronized before they can be used in the receiving domain [6], [7].

A GALS NoC implementation can take many forms. The most flexible one would enable CDC in any part of the NoC [8]. In this case, the NoC itself would be composed of different sectors operating in different clock domains. This is achieved by allowing any point-to-point link in the NoC to belong to different clock domains.

The physical implementation scalability of NoCs should not be limited by the architectural features that they need to support to provide high communication performance and Quality-of-Service (QoS) guarantees. To achieve these goals, NoCs typically employ Virtual Channels (VCs) that allow for traffic separation (isolation) by assigning different traffic classes to different VCs [9], [10], [11]. 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 [12]. Also, VCs are used to break channel dependencies and avoid network-level deadlocks under adaptive routing [13].

In this work, our goal is to highlight the intricacies of mesochronous CDC on links with VC flow control and to introduce a new efficient architecture that would reduce the overall hardware cost without sacrifycing throughput. The proposed approach, called ***FastCross***, leverages existing mesochronous synchronizers to provide a loosely-coupled communication across mesochronous clock domains. In this way, VC buffer sharing can be safely applied and neighboring interfaces can be consolidated to jointly synchronize data and flow-control signals transferred across neighbor NoC components.

II. VC flow control on mesochronous links

Transferring data safely across two mesochronous clock domains can be performed with various techniques.

A. Mesochronous Synchronizer

The most scalable approach relies on the "n-flop" synchronizer [5], [14], [15], as shown in Fig. 1. The n-flop synchronizer consists of n registers placed in parallel in the transmitter domain ($n = 4$ in Fig. 1) and two free-running counters that are monotonically incremented in every cycle.

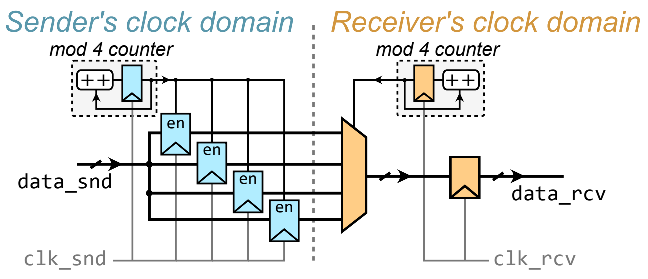

Fig. 1. The n-flop mesochronous synchronizer, for $n = 4$.

In the forward direction, each flit is written to one of the four registers of the mesochronous synchronizer. The register that is activated for each write is selected by the write pointer of the mesochronous synchronizer. Both the data registers and the write pointer are driven by the sender's clock.

On the receiver's side, a new word is read out of the synchronizer every cycle of the read domain. The register that is being read is selected by the read pointer, which is driven by the receiver's clock.

The read and write pointers are free-running counters (modulo 4), which get incremented on every positive edge of their clock. The increments occur irrespective of the validity of the data that gets written to, or read out of, the mesochronous synchronizer. For safe crossing between the two mesochronous clock domains, the values of the two pointers always differ by two. This difference is guaranteed during the reset phase of the synchronizer [16].

B. Tightly coupled VC Flow-Control across Clock Domains

To divide a physical channel into V VCs, the receiving side of the link should provide as many independent queues as the number of VCs. Each VC is independently flow controlled thus disallowing any dependencies between VCs to emerge.

To support VC flow control across mesochronous clock domains can be done either by employing one synchronizer per VC at the receiver's side [17], [18] or by using a dual-clock mesochronous queue [19] per VC. In this case, the transferred data is written directly to the appropriate VC, assuming that the corresponding queue is not full. In both cases, synchronization and flow-control are *tightly coupled* and closely integrated.

While the aforementioned approach may seem attractive due to its low complexity, there are significant arguments against its use: (1) it is not advisable to incorporate numerous CDC points on a single clock boundary (i.e., one such point per VC), because both the probability of failure and the needed verification effort increase markedly; (2) by statically partitioning the VC buffers, one precludes the option of VC buffer sharing, which is a technique frequently employed by state-of-the-art NoC designs to maximize buffer utilization; (3) the negative impact on hardware cost may be dramatic, because each VC buffer must be sufficiently deep to store all in-transmission data – as dictated by the round-trip delay – to achieve full-throughput performance.

C. Loosely Coupled VC flow-Control Synchronization

To minimize the number of independent CDC interfaces is to abandon the *tightly coupled* approach and employ a *loosely coupled* setup where synchronization and buffering are separated as shown in Fig. 2. In this case, the synchronizer merely acts as a delay element from a flow-control perspective. This decoupling allows us to adopt a credit-based VC flow control and buffer sharing.

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 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.

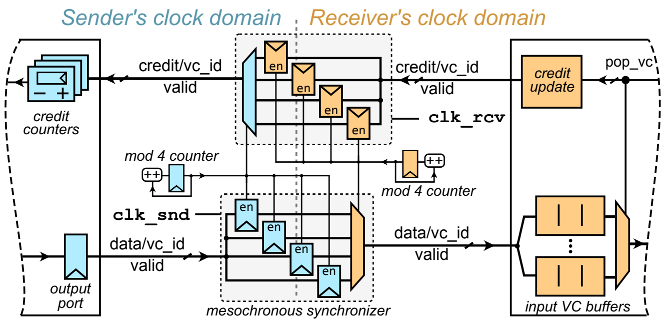

Fig. 2. FastCross interfaces connecting mesochronous clock domains. Forward data and backward credit updates are separately synchronised using 4-deep mesochronous synchronizers. The read/write pointers of the two synchronizers are shared in both directions.

In the forward direction, each flit – together with the VC ID that describes the VC the flit belongs to, and a validity bit – are written to one of the four registers of the mesochronous synchronizer. The register that is activated for each write is selected by the write pointer of the mesochronous synchronizer. Both the data registers and the write pointer are driven by the sender's clock. The corresponding flit can leave the sender and be written into the synchronizer, as long as the selected VC has at least one credit available, which is consumed while the flit is still on the sender's side.

On the receiver's side, a new word is read out of the synchronizer every cycle of the read domain. If the word that is read out of the synchronizer is valid, it is written into the corresponding VC buffer (recall that a credit has been consumed at the sender, before the word is put in the synchronizer). When the word is invalid, it is ignored. In both cases, the register that is being read is selected by the read pointer, which is driven by the receiver's clock.

In the opposite direction, from where the produced credits return to the sender, a similar structure is applied. A new credit update – together with a credit validity bit – are written to one of the four registers of the credit synchronizer, and, after two cycles, they are read out on the sender's side. The valid credit updates increase the available credits on the sender's side, while the invalid ones are ignored.

Instead of using separate pointers for the credit mesochronous synchronizer, we *reuse* the free-running pointers of the synchronizer of the forward direction. Thus, credit updates are written to the register of the synchronizer pointed by the read pointer, and they are read out on the sender's side via the read multiplexer (using the write pointer). As long as both counters never stop counting and they keep a constant difference of 2 (modulo 4), they fulfill the requirements of proper mesochronous synchronization.

D. Interface Consolidation

In the case of bidirectional interfaces, whereby two NoC components are connected with a pair of uni-directional links (allowing both sides to send and receive data), the number of CDC points are effectively doubled.

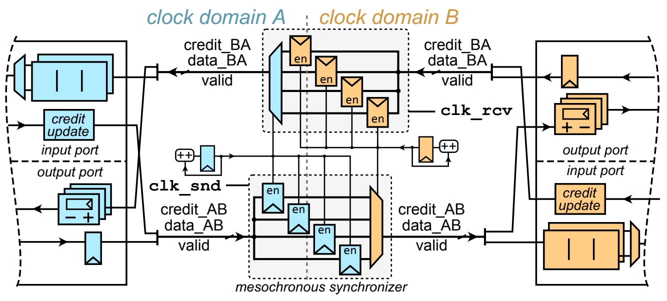

Fig. 3. Consolidated mesochronous FastCross interfaces that enable a single CDC point for both data and credits.

For such cases, we propose the consolidation of the synchronization interfaces for maximizing design safety and simplifying verification. Consolidation means that the data and credits that flow from router A to B (and vice versa) are merged and synchronized at one entry point at the inputs of B (and A, respectively) (see Figure 3). The separate synchronizers used for the data and credits in the case of unidirectional links, are merged with the synchronizers of the opposite direction. Each entry of the consolidated synchronizer can carry a new flit, or a returning credit, or both, simultaneously. With this approach, FastCross reduces the CDC points *in each direction* to merely *one*, thereby simplifying verification and increasing the system's reliability.

The pointers that govern the synchronization are shared across both sides. This sharing of read and write counters of mesochronous synchronizers for both directions further increases also the reliability of data transfer. Mesochronous synchronizers face metastability only during reset, when the reset signals must be synchronized using brute-force synchronizers to both domains. This reset signal actually provides the necessary separation for the read and write pointers, which allows signals to safely pass from one domain to the other. Since the need to separately reset multiple independent pointers is removed when using FastCross, the probability of experiencing concurrent timing failures during reset at multiple independent CDC points is minimized.

III. Evaluation

With FastCross, the overall delay of the forward path to write data into the VC buffers of the receiver is 3 cycles: one cycle is spent in writing new data into the mesochronous synchronizer; the data will then be read out of the synchronizer after 2 cycles, due to the constant 2-cycle separation of the read and write pointers. In the same cycle that the data is read out of the synchronizer, the data is placed in the VC buffers of the receiver. Equivalently, the delay of credits returning to the sender is 3 more cycles: once a new credit update is produced, it needs one cycle to be written into the credit synchronizer. After 2 cycles, the credit update will be available to the sender, which can reuse it immediately. Thus, the overall round-trip delay is equal to 6 cycles.

The number of buffer slots per VC and the corresponding credits should be able to cover this round-trip delay, in order to enable 100% throughput. This is achieved by budgeting 6 buffer slots per VC. The total number of buffers can be significantly reduced by employing VC buffer sharing.

To quantify the buffer savings enabled by FastCross, we compare it with a tightly coupled organization that assumes one independent mesochronous FIFO queue per VC [17]. In such cases, the arrival of the flow control signals is delayed by two cycles, due to the constant 2-cycle separation of the write and read pointers inside each mesochronous FIFO. This delay determines the size of the FIFO buffers to 4 slots (to enable 100% of throughput).

The third architecture under comparison is LIME [18], a native VC-based mesochronous link architecture that operates under credit-based flow control. LIME reduces the synchronization overhead by only using a dual-stage synchronizer for the flow-control signals, i.e., the forward valid signals and returned credits. Data moves from the sender directly to the receiver, without any synchronization element in-between. In order to avoid timing failures, the FIFO cannot be read directly after the write operation, but only after the dual-stage synchronizer of the forward flow-control signals permits it, i.e., two cycles later. In order to achieve 100% throughput operation, LIME requires 1 less buffer slot per VC, as compared to FastCross. However, LIME *cannot* leverage shared buffering, since it relies on separate FIFOs per VC.

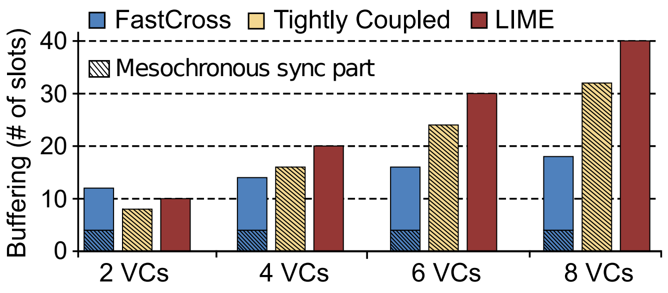

Fig. 4. The buffering required to allow 100% throughput in 3 different mesochronous VC flow-controlled architectures: (a) the "FastCross" mesochronous architecture, (b) a valid/stall-based tightly coupled mesochronous sychronizer ("Tightly Coupled"), and (c) the LIME architecture.

The minimum buffering requirements for all three architectures under comparison are depicted in Figure 4. Both the "LIME" and the "Tightly Coupled" architectures make the most of the available buffering over a 2-VC mesochronous link. However, when more VCs are employed, FastCross dominates with up to 55% buffer reduction.

A similar trend is observed in Figure 5, which plots the hardware-area profile of all three architectures. All three designs were implemented in SystemVerilog and synthesized using a 45 nm standard-cell library (at 0.8 V, 125 ^{o}C). All configurations use 64-bit-wide data words, and both the sender and receiver clocks are constrained to 1 GHz in all cases. Note that "FastCross" uses a slightly wider data synchronizer, in order to fit the credit for the opposite direction. Shared buffering is implemented using the ElastiStore architecture [20].

In the 4-VC configuration, "FastCross" ends up occupying the same area as the "Tightly Coupled" architecture (despite requiring less buffering, as indicated in Figure 4). This is the result of the more complex shared buffer. Nevertheless, under the 6- and 8-VC configurations, FastCross achieves 20% and 34% lower area than the "Tightly Coupled" design, respectively.

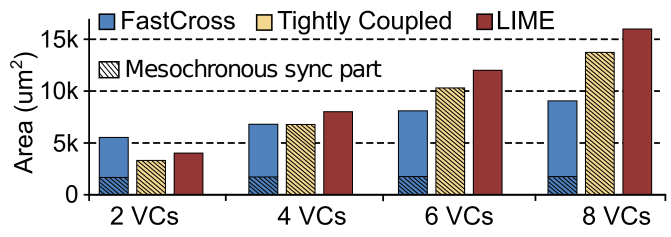

Fig. 5. Hardware area occupied by 3 different mesochronous VC flow-controlled architectures: (a) the "FastCross" mesochronous architecture, (b) a valid/stall-based tightly coupled mesochronous sychronizer ("Tightly Coupled"), and (c) the LIME architecture [18].

The concept of *tightly coupled* mesochronous approaches [21] – whereby the link synchronization operation is integrated with input buffering and flow control – is effective in making efficient use of the hardware resources *when the number of employed VCs is relatively small*. This is, indeed, verified by hardware results in the case of NoCs with 2 VCs, as seen in Figure 5. The same conclusion can be made for LIME [18]. Although LIME does not strictly follow the *tightly coupled* approach, it still tackles the synchronization overhead by "tightly" combining input buffering and synchronization operations. However, this approach becomes inefficient in multi-VC NoCs. On the other hand, FastCross decouples synchronization from the VC flow-control semantics. This decoupling enables the use of shared buffering at the receiver, which pays off – in terms of hardware cost – as the number of VCs increases. Thus, the *loosely coupled* FastCross approach scales much more efficiently with the number of VCs.

IV. Conclusions

This article has identified and analyzed the intricacies involved in the application of VC flow control across mesochronous clock domains. Contrary to state-of-the-art approaches that focus on tightly-coupled mesochronous interfaces, we have shown that loosely coupled approaches that distinguish synchronization from VC buffering are more beneficial for mesochronous interfaces. The sharing of VC buffers and consolidation of the interfaces amortizes the cost of using separate synchronizers in front of the VC buffers, for increased number of VCs.

References

[1] P. Teehan, M. Greenstreet, and G. Lemieux, "A survey and taxonomy of gals design styles," *IEEE Design Test of Computers*, vol. 24, no. 5, pp. 418–428, Sept 2007.

[2] W. J. Dally, C. Malachowsky, and S. W. Keckler, "21st century digital design tools," in *Proc. of DAC*, 2013.

[3] G.Campobello and et al., "GALS networks on chip: a new solution for asynchronous delay-insensitive links," in *DATE*, 2006, pp. 160–165.

[4] A. Psarras, M. Paschou, C. Nicopoulos, and G. Dimitrakopoulos, "A dual-clock multiple-queue shared buffer," *IEEE Trans. on Computers*, vol. 66, no. 10, pp. 1809–1815, 2017.

[5] W. J. Dally and J. W. Poulton, *Digital systems engineering*. Cambridge University Press, 2008.

[6] T. Chelcea and S. M. Nowick, "Robust interfaces for mixed-timing systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 8, pp. 857–873, Aug 2004.

[7] R. Ginosar, "Metastability and synchronizers: A tutorial," *IEEE Design & Test of Computers*, vol. 28, no. 5, pp. 23–35, 2011.

[8] P. Bouchard, P. Martin, and J.-J. Lecler, "Network-on-chip (noc) with qos features," 2012, uS Patent 2013/0179613A1.

[9] W. J. Dally, "Virtual-Channel Flow Control," in *Int. Symp. on Comp. Arch. (ISCA)*, 1990, pp. 60–68.

[10] A. Psarras, J. Lee, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "PhaseNoC: Versatile network traffic isolation through tdm-scheduled virtual channels," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 844–857, 2015.

[11] I. Seitanidis, A. Psarras, E. Kalligeros, C. Nicopoulos, and G. Dimitrakopoulos, "ElastiNoC: A self-testable distributed VC-based network-on-chip architecture," in *IEEE Int. Symp.on Networks-on-Chip (NoCS)*, 2014, pp. 135–142.

[12] M. Martin, "Token coherence," Ph.D. dissertation, Univ. of. Wisconsin, 2003.

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

[14] A. Edman and C. Svensson, "Timing closure through a globally synchronous, timing partitioned design methodology," in *Proc. of the Design Automation Conference (DAC)*, 2004, pp. 71–74.

[15] M. Ghoneima, Y. Ismail, M. Khellah, and V. De, "Variation-tolerant and low-power source-synchronous multicycle on-chip interconnect scheme," *Hindawi VLSI Design*, 2007.

[16] D. Verbitsky and et al., "Starsync: An extendable standard-cell mesochronous synchronizer," *Integration*, no. 2, pp. 250–260, 2014.

[17] 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.

[18] 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.

[19] D. Konstantinou, A. Psarras, C. Nicopoulos, and G. Dimitrakopoulos, "The mesochronous dual-clock FIFO buffer," *IEEE Transactions on VLSI Systems*, vol. 28, no. 1, pp. 302–306, 2019.

[20] I. Seitanidis, A. Psarras, K. Chrysanthou, C. Nicopoulos, and G. Dimitrakopoulos, "Elastistore: Flexible elastic buffering for virtual-channel-based networks on chip," *IEEE Transactions on VLSI Systems*, vol. 23, no. 12, pp. 3015–3028, 2015.

[21] D. Ludovici, A. Strano, D. Bertozzi, L. Benini, and G. N. Gaydadjiev, "Comparing tightly and loosely coupled mesochronous synchronizers in a noc switch architecture," in *ACM/IEEE International Symposium on Networks-on-Chip*, May 2009, pp. 244–249.