The Mesochronous Dual-Clock FIFO Buffer

Dimitrios Konstantinou, Anastasios Psarras, Chrysostomos Nicopoulos, and Giorgos Dimitrakopoulos

***Abstract*— To increase system composability and facilitate timing closure, fully synchronous clocking is replaced by more relaxed clocking schemes, such as mesochronous clocking. Under this regime, the modules at the two ends of a mesochronous interface receive the same clock signal, thus operating under the same clock frequency, but the edges of the arriving clock signals may exhibit an unknown phase relationship. In such cases, clock synchronization is needed when sending data across modules. In this brief, we present a novel mesochronous dual-clock first-input–first-output (FIFO) buffer that can handle both clock synchronization and temporary data storage, by synchronizing data implicitly through the explicit synchronization of only the flow-control signals. The proposed design can operate correctly even when the transmitter and the receiver are separated by a long link whose delay cannot fit within the target operating frequency. In such scenarios, the proposed mesochronous FIFO can be extended to support multicycle link delays in a modular manner and with minimal modifications to the baseline architecture. When compared with the other state-of-the-art dual-clock mesochronous FIFO designs, the new architecture is demonstrated to yield a substantially lower cost implementation.**

***Index Terms*— Clock-domain crossing, mesochronous first-input–first-output (FIFO), source-synchronous communication.**

I. Introduction

Modern systems-on-chip (SoC) implemented in deeply scaled technologies faces 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 [1], [2]. Partitioning the SoC into globally asynchronous, locally synchronous domains [3], [4] partially alleviate the problem, since synchronous operation and its associated timing constraints are confined inside each domain. However, in this case, when crossing clock domains, the signals must be synchronized to the receiving clock domain, in order to avoid metastability [5], [6]. In addition to delivering synchronized signals across the clock domain interface, it is also important to ensure that any synchronized data that cannot be immediately consumed by the receiving domain are safely stored until it can be serviced. Since data must both be synchronized and (temporarily) stored, it is imperative that these two elemental and intertwined operations—synchronization and buffering—are combined in a cost-effective way that minimizes any latency and area overhead.

In this brief, we focus on mesochronous clock domains, where clocks operate under the same frequency, but with a fixed, arbitrary phase difference. In such cases, using a generic asynchronous dual-clock first-input–first-output (FIFO) [7] for mesochronous clock domain, crossing is possible, but incurs unnecessary latency overhead [8]. Currently, there are two major approaches for efficient synchronization and buffering across mesochronous interfaces: 1) in a loosely coupled implementation [8], synchronization and buffering occur separately while 2) in a tightly coupled implementation [9]–[11], they are combined and fused into a single structure. While the former approach allows for maximum flexibility in terms of system composition and can readily support multicycle link delays, it incurs large area overhead and exhibits significant resource underutilization. On the other hand, the latter approaches maximize resource utilization and minimize the area overhead, but they operate only under single-cycle timing constraints, or require phase detectors and cannot be generalized to links with multicycle delays.

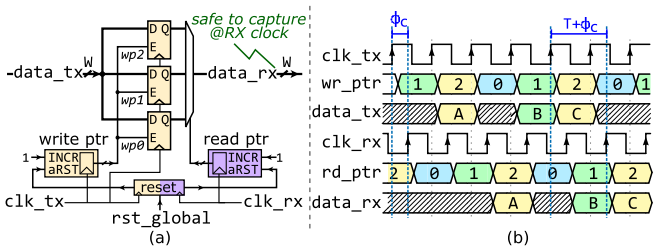

Fig. 1. (a) Three-flop synchronizer used for mesochronous clock-domain crossing and (b) example of its operation. The spread between the write and read pointer free-running counters guarantees that `data_tx` is always read out after a safe time period has elapsed since it was written.

In this article, the ultimate objective is to combine the benefits of both loosely and tightly coupled approaches. Specifically, a novel buffering structure that integrates mesochronous synchronization is proposed, which successfully combines area efficiency, high performance, and support for multicycle link delays. The new mesochronous dual-clock FIFO architecture is the first (to the best of our knowledge) to achieve data synchronization implicitly, through the explicit synchronization of only the flow-control signals. This radical new approach is instrumental in simplifying the overall design, since it obviates the need to synchronize physically the—typically wide—data buses. Due to its optimized organization, the proposed synchronizer provides three key benefits: 1) it is the first architecture to achieve lossless operation under any buffering depth ≥ 1; 2) it imposes minimal buffering requirements for the full-throughput operation; and 3) it can be seamlessly extended to support multicycle links without any further restrictions. Extensive and detailed hardware evaluation validates the efficacy and efficiency of the new design. The proposed architecture is demonstrated to outperform the two most relevant state-of-the-art existing solutions in all salient design metrics.

II. Mesochronous Synchronization and Buffering

Transferring data safely across two mesochronous clock domains can be performed with various techniques. The most scalable approach relies on the "n-flop" mesochronous synchronizer [5], [12], [13], as shown in Fig. 1(a). The n-flop synchronizer consists of n registers placed in parallel in the transmitter domain ($n = 3$ in Fig. 1) and two free-running counters that are monotonically incremented in every cycle. The two counters are placed on opposite domains and control the position among the n flops where the synchronizing signal is stored and read out. A reset synchronization structure initializes the counters to their starting values.

Manuscript received June 25, 2019; revised August 29, 2019 and September 23, 2019; accepted October 3, 2019. *(Corresponding author: Giorgos Dimitrakopoulos.)*

D. Konstantinou, A. Psarras, and G. Dimitrakopoulos are with the Electrical and Computer Engineering Department, Democritus University of Thrace, 67100 Xanthi, Greece (e-mail: dkonstan@ee.duth.gr; apsarra@ee.duth.gr; dimitrak@ee.duth.gr).

C. Nicopoulos is with the Electrical and Computer Engineering Department, University of Cyprus, Nicosia 1678, Cyprus (e-mail: nicopoulos@ucy.ac.cy).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2946348

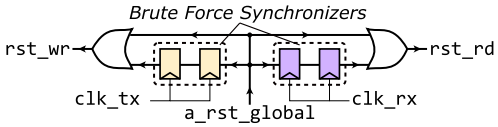

Fig. 2. Asynchronous reset synchronization structure used to initialize the counters of the n-flop synchronizer.

Metastability-free operation relies on the fact that the signal is not read out by the receiver unless a safe time period has elapsed, since the transmitter has written to that register. This is guaranteed by imposing a so-called spread between the values of the free-running counters. As highlighted in the transfer of word "B" in the running example of Fig. 1(b), if a register is written at time t by the transmitter clock, it will be captured by the receiver registers at time $t+T+\phi_c$, where T is the clock period and ϕ_c is the TX/RX clock phase skew. In most phase-skew scenarios, $n=2$ flops suffice to provide this guarantee [14]. However, as indicated in [5], if the two clocks are almost in phase (i.e., $\phi_c \approx 0$ or $\phi_c \approx T$), then a 2-flop synchronizer cannot guarantee that the output data will remain stable throughout a whole receiver clock cycle. Therefore, for safe operation under any skew phase, and without using a phase detector as done in [11] and [15], a 3-flop synchronizer is required.

A 3-flop synchronizer suffices when a fully deterministic reset process as the one presented in [12] is employed. In this case, during reset, the clocks are disabled and reenabled only after reset is deasserted and all registers received their starting values. Although this reset procedure sounds appealing due to its fully deterministic behavior, in practice, it requires a systemwide coordination of clock and reset signals that minimize flexibility in system composition.

To overcome this deficit, the initialization process of the state-of-the-art mesochronous synchronizers [8] is performed with an asynchronous reset signal that is synchronized independently to both the sender and receiver domains. This synchronization of the asynchronous reset signal to both domains is performed with the use of dual brute-force synchronizer structures, as shown in Fig. 2. Depending on the reliability requirements, the brute-force synchronizers can be implemented with more in-series flip-flops. In any case, it is possible for any of the two counter reset signals to enter metastability and settle to their deasserted value with one cycle delay. In [8], it was shown that to account for this possible metastability during the (write and read) counters' initialization phase and to ensure that a minimum spread of at least one is achieved after reset, we need a 4-flop synchronizer to ensure correct operation, irrespective of the phase difference between the two clocks.

Due to the free-running property of the n-flop synchronizer's write/read pointer counters, synchronized data must be read out in each cycle by the receiver. In reality, however, it is possible that data cannot be consumed immediately [16], [17]. In order to temporarily store synchronized data, Starsync [8] places a synchronous FIFO buffer right after the data synchronizer, as shown in Fig. 3(a). A push/full flow control is employed on the mesochronous interface, so that the transmitter stops sending data when the receiver's buffers are full. The backward `full` signal indicates when the buffer is full; the forward transmitter-driven `push` signal indicates whether the incoming data are valid. Both signals are synchronized when crossing the mesochronous interface using separate single-bit 4-flop synchronizers, as required to cover the worst case scenario for asynchronous reset initialization.

A different organization is proposed in [9], which implements buffering within the synchronizer, as shown in Fig. 3(b). In this tightly coupled implementation, no extra buffering is employed; instead, the 3-flop synchronizer in Fig. 1(a) also acts as a buffer by relaxing the counters' free-running characteristic. Whenever synchronized data

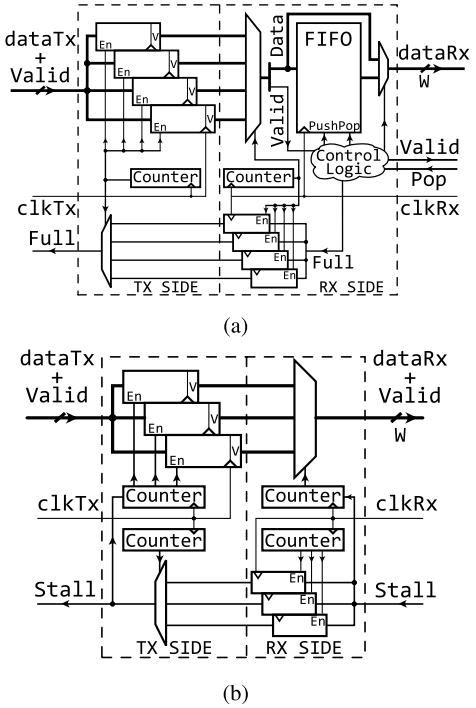

Fig. 3. (a) Loosely coupled [8] and (b) tightly coupled [9] mesochronous synchronization and buffering architectures.

cannot be consumed immediately in the receiver due to a downstream stall, the write and read counters stop and the transmitter is notified by a `stall` signal to stop sending further data.

The tightly coupled organization reduces the area budget, since no extra registers are required to implement the FIFO buffer, other than the ones required by the modified 3-flop synchronizer. However, in order to stop the free-running counters effectively, the stall signal must reach the transmitter within the same clock cycle. This requirement imposes very strict timing constraints to the transmitter–receiver link, thereby limiting the maximum link length that can be supported. In order to somewhat relax the timing constraints, an alternative "hybrid" version of the tightly coupled organization was proposed [10]. In both cases, 3-flop synchronizers are enough, since in [9] and [10], the clocks are assumed to be deasserted during reset, similar to [12]. Instead, if reset is done asynchronously, as in Starsync [8], a 4-flop synchronizer will be needed.

Overall, the sizing of the synchronizer structures in both loosely and tightly coupled architectures is not an architectural feature, but just a property of the employed reset mechanism. Therefore, even though the two architectures were proposed with different register requirements, in reality, they need the same n-flop synchronizers, as long as they are being reset in the same way.

III. Proposed Mesochronous FIFO

The proposed mesochronous FIFO architecture combines the benefits of the loosely coupled [8] and tightly coupled approaches [9], [10], while avoiding their weaknesses. The new design couples synchronization and buffering in a cost-efficient implementation that fully supports multicycle link delays. A completely different operating approach is adopted, whereby the data are synchronized implicitly through the explicit synchronization of flow-control signals.

A. Architecture and Operation

Fig. 4(a) shows the proposed mesochronous FIFO. Data that need to be synchronized are stored in a memory placed in the transmitter domain. Two monotonically increasing counters

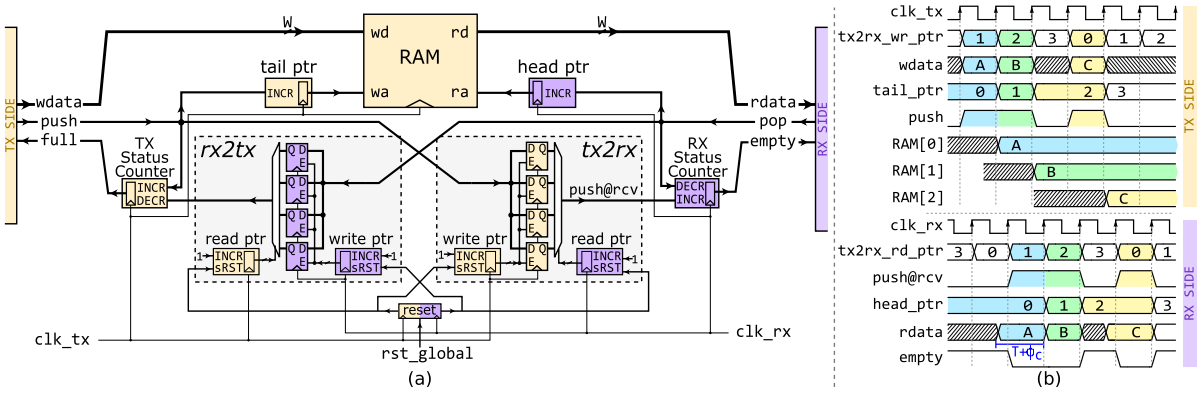

Fig. 4. (a) Proposed mesochronous FIFO, which implicitly synchronizes data through the explicit synchronization of the flow-control push/pop signals. (b) Short cycle-by-cycle example of the operation of the proposed design.

index the memory positions, where data are stored and accessed. The transmitter-synchronous tail pointer points to the write position in memory where a new data word is stored, while the receiver-synchronous head pointer points to the position from where a word will be read out. A pair of opposite-direction single-bit 4-flop synchronizers are used to sync enqueue (write) and dequeue (consume) events between the two sides. The 4-flop synchronizers are used to account for the worst case scenario that can occur with the asynchronous reset of the read and write pointers, as described in [8].

When the transmitter sends a data word that needs to be synchronized, it writes the data into the memory position pointed by the tail pointer. At the same time, the tail pointer is increased, and the push signal that is fed to the forward "tx2rx" mesochronous synchronizer [see Fig. 4(a)] is asserted. When the enqueue event is synchronized across the interface, the receiver can safely read out the data from the memory position pointed by the head pointer. This operation is shown in the short example of Fig. 4(b), which depicts the transfer of three data words ("A," "B," and "C") from TX to RX. Once the receiver actually consumes the data (e.g., reads it and sends it downstream), the `pop` signal is asserted, and the head pointer is incremented to point to the position where the next data word is found. Note that the receiver does not try to read data from the updated head pointer position, unless a new push event has been synchronized, indicating that new data exist and are safe to be read out. In order to not lose track of multiple enqueue events, the receiver employs a "status" counter that counts the number of synchronized data items currently in the queue. Whenever a new data word is received, as indicated by an incoming enqueue signal from the "tx2rx" synchronizer in Fig. 4(a), the counter is increased; when a data word is consumed, the counter is decreased to reflect the change in the queue's state. In this way, two key objectives are achieved: 1) data are implicitly synchronized through the explicit synchronization of the enqueue events and 2) the FIFO order is preserved in the buffer.

The next step is to synchronize the queue's state to the transmitter domain and guarantee that the queue does not overflow. To achieve this, the transmitter also uses a status counter, as shown in Fig. 4(a), to keep its own version of the number of items currently stored in the queue. The counter is incremented, or decremented, whenever an item is enqueued, or dequeued, from the queue, respectively. Since dequeue (pop) events are receiver-synchronous, they have to be synchronized to the transmitter domain through a separate backward synchronizer. On a dequeue, the receiver asserts the `pop` signal of the "rx2tx" synchronizer. Once the signal is synchronized, the transmitter decreases its status counter, effectively remaining in sync with the downstream buffer's state.

Similar to [8], the forward latency of synchronizing and enqueuing a new data item is between one and three cycles, depending on the spread between the read and write pointers after reset. For safe operation under any phase difference, the initial spread between the read and write pointers at each synchronizer is two. When the reset of the read pointer of the push synchronizer is delayed (due to metastability), the forward latency is increased by one cycle. On the contrary, if the reset of the write pointer is delayed, the forward latency is decreased. In the case that the reset signal is not delayed, or delayed on both sides, the forward latency remains unchanged. The backward latency, i.e., the number of cycles needed to synchronize the pop events, is also between one and three cycles. However, worst case forward and backward latencies cannot occur simultaneously. The read pointer of the push synchronizer is driven by the same reset signal (the rx-side-synchronized version of the asynchronous reset) with the write pointer of the pop synchronizer. The same happens with the write and read pointers that are driven by the reset signal synchronized to the tx side. Therefore, once one side experiences a latency of three cycles (delayed reset of the read pointer), the other side will experience a latency of one cycle (delayed reset of the write pointer). Overall, the sum of the forward and backward latencies is constant at four cycles.

B. Support for Multicycle Source-Synchronous Links

The baseline mesochronous FIFO architecture in Fig. 4(a) assumes that the transmitter is placed within close physical proximity to the receiver, and thus, no timing problems will be introduced due to wire delays between them. When this assumption cannot hold, e.g., when the transmitter and the receiver are separated by a long physical link whose delay cannot fit within the target operating frequency, the proposed mesochronous FIFO can be extended to support multicycle link delays in a modular manner and with minimal modifications to the baseline architecture.

The reorganized multicycle architecture is shown in Fig. 5, assuming R_F forward and R_B backward register stages. We implement the multicycle link by inserting as many register stages as required to cover the desired timing constraint. Both forward (data, push) and backward (pop) signals are captured by registers clocked by the transmitter clock. A source-synchronous clock (strobe) is sent out by the transmitter to clock the registers on each path.

Extending the baseline (single-cycle) architecture to support multicycle delays requires a careful selection of the split point, i.e., the point where the multicycle delay path will be inserted within the FIFO architecture, and the consequent reorganization of its internal logic, as required by the selected split-point placement. Although there are multiple alternatives that can achieve correct FIFO functionality, only few of them can achieve minimum implementation cost under optimal performance. One of the straightforward choices is the placement

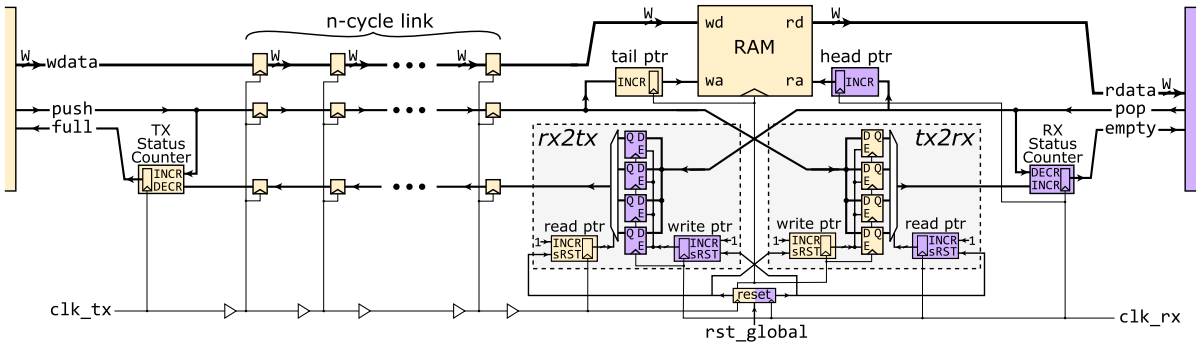

Fig. 5. Reorganization of the proposed mesochronous FIFO when the length of the link between the transmitter and the receiver is too long to fit within a single cycle. In such cases, the link is split into multiple register stages, for both forward and backward data and flow-control signals.

of the memory and the head pointer on the receiver side, so that, whenever a word is consumed, the next data item can be retrieved immediately in the following cycle. If the memory and head pointer were placed before the multicycle link, then the next data item would reach the receiver with an $(R_B + R_F)$-cycle delay: R_B cycles for the pop event to reach the head pointer and increment it and R_F cycles for the next data word to reach the receiver.

Another critical, but less straightforward, design decision is the placement of logic that produces the empty and full states of the FIFO. We choose to place the RX and TX status counters—that produce the empty and full states of the FIFO, respectively—on the receiver and transmitter sides, respectively. This choice is the critical difference of our implementation, as compared with the multicycle link implementation of the loosely coupled StarSync architecture [8], which places the full-state calculation logic on the receiver side and then transfers this state to the transmitter. As a result, in StarSync [8], the calculation of the full state must be modified to account for the $R_F + R_B$ data words that may arrive by the time the full signal has reached the transmitter: R_F that might already be in-flight on the forward link and R_B more that may be sent in the time window between the assertion of the full signal and its reception by the transmitter [7], [18]. In addition to introducing buffer underutilization due to early throttling (before the FIFO has actually become full), this choice also incurs increased costs by setting a minimum buffering requirement for lossless operation. Instead, the proposed approach of transmitting and synchronizing push and pop events allows the state to be calculated locally on each side, without considering possible in-flight data. This leads to optimal buffer utilization; even with an FIFO depth of a single slot, lossless transmission is still guaranteed.

IV. Analysis and Experimental Results

In this section, we evaluate the proposed mesochronous FIFO and compare it against the two state-of-the-art alternatives that combine synchronization and data buffering [8], [9], as described in Section II. All designs are implemented in SystemVerilog and their hardware costs are evaluated after synthesizing the hardware models in a 45-nm standard-cell library at 0.8 V and performing placement-and-routing of the resulting designs using the Cadence backend flow.

First, we study the hardware implementation efficiency of a mesochronous interface where the transmitter and the receiver are placed close to each other, separated by a link with a delay that can fit in a single clock cycle. In all designs under comparison, we assume that the read and write pointers of the synchronizers are initialized using the generic asynchronous reset synchronizers proposed in Starsync [8] and highlighted in Fig. 2. Under this scenario, every n-flop synchronizer should consist of $n = 4$ flops. The state-of-the-art mesochronous synchronizers and buffers [8], [9] use 4-flop synchronizers directly on the data lines. Therefore, each stage of the synchronizer involves registers equal to the data width. On the contrary, the proposed design uses single-bit 4-flop synchronizers that are used to synchronize only the push and pop control signals.

For the design of [9], the wide 4-flop synchronizer is enough to perform synchronization and buffering. However, for Starsync [8] and the proposed design, additional data registers are required to implement the FIFO storage. In both cases, the minimum FIFO depth required to deliver full-throughput operation under any data-burst length is determined by the round-trip time (RTT). Assuming that the transmitter pushes a data word at time t, the receiver can consume it—and, thus, free its corresponding memory position—once the push event has been synchronized at time $t+$`FwdLatency`. However, the transmitter will be notified only after the pop event has been synchronized on its side, at time $t+$`FwdLatency`$+$`BwdLatency`. The sum of the forward and the backward latency is determined by the initial spreading of the write and read free-running counters of each n-flop synchronizer, and it is always equal to four cycles, as previously discussed for the proposed design. The worst case latency of three cycles cannot occur in both forward and backward synchronizers, as long as the pointers of each synchronizer receive the same local synchronized version of the asynchronous global reset. Therefore, for maximum throughput, under asynchronous reset, StarSync [8] and the proposed design require 4-slot FIFOs.

The top part of Table I reports the area and power of each design for 4-slot deep structures obtained at a target frequency of 1.5 GHz and assuming a data width of 64 bits. Even though the proposed design is architected to operate under both single- and multicycle links, it requires only marginally more area than the tightly coupled design of [9] that is tailored only for single-cycle links. Despite this small area overhead, the proposed design is as power-efficient as the design of [9] for high data traffic rates and significantly more power-efficient at lower data traffic rates. This attribute is revealed by the power consumption reported in Table I for three different data injection traffic rates for all designs under comparison. At low (10%) and medium (50%) traffic, the proposed design requires around 58% and 20%, respectively, less power than the design in [9]. At maximum traffic (100%), the power of both designs is almost identical, since they both synchronize valid data on every cycle.

In the absence of an explicit stall signal from the receiver, the Tight [9] architecture exhibits switching activity, even when it is idle (i.e., when it does not have any valid data to send to the receiver). This is the reason why its power consumption does not drop significantly at 10% injection traffic rate. Of course, on an output stall, all synchronizers under comparison exhibit zero output data switching activity.

As expected, the increased buffer requirement of the loosely coupled approach [8] is much more prominent than the tightly coupled approach [9], and it effectively doubles the area

TABLE I

HARDWARE IMPLEMENTATION RESULTS OF THE THREE SYNCHRONIZERS FOR ZERO-CYCLE LINK DELAY AT 45-nm TECHNOLOGY AND 0.8 V

slots	Design @1.5 GHz	Area (μm^2)	Power(uW) @rate% 10%	50%	100%	Max. Freq GHz
4	Tight [9]	1699	108	167	241	3.01
	Starsync [8]	3599	343	410	435	2.15
	Proposed	1793	44	132	241	2.88
3	Tight [9]	1279	80	126	183	3.10
	Starsync [8]	2643	237	305	331	2.32
	Proposed	1347	34	101	184	2.92

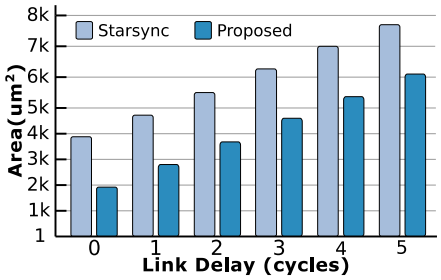

Fig. 6. Area occupied by the synchronizers as the delay of the link increases. In this comparison, the tightly coupled approach [9] is not included, because it does not support multicycle link delays.

requirements. Consequently, the extra FIFO data buffer and the free-running synchronizer that accommodates entire data words (instead of merely single-bit signals, as in the proposed design) significantly increase the area and power consumption.

For completeness, the bottom part of Table I compares the same designs using 3-flop synchronizers and FIFO buffers, as originally proposed in [9], which assumes that clocks can be fully disabled during reset, as proposed in [12]. When using 3-flop synchronizers, the proposed design enjoys the scalability of Starsync [8] with the area requirements of the restricted tightly coupled design [9]. Again, power consumption follows the same trend: the proposed design offers significant power savings at lower throughput, which diminish at maximum-throughput operation. Finally, note that the proposed design enables lossless operation even with a shallower FIFO (fewer than three slots), but at a lower throughput. This tradeoff is not an option for the other state-of-the-art architectures.

The last column of Table I depicts the maximum frequency that can be achieved by each design. Tight [9] is marginally faster than the proposed design and significantly faster than Starsync [8]. However, this extra speed benefit comes at a high cost: the lack of flexibility that inhibits the design of [9] from being integrated into the frequently encountered multicycle clock-domain crossing interfaces. Thus, for the case of multicycle links, only Starsync [8] and the proposed design are compared. Their area requirements pertaining to multicycle link delays are highlighted in Fig. 6.

Due to its loosely coupled nature and because it employs a 4-flop synchronizer directly on the input data, Starsync [8] always needs four datawide registers for the synchronizer outside of the FIFO buffer. The 4-flop synchronizer cannot be merged with the subsequent FIFO structure. Instead, the proposed architecture removes this overhead by avoiding the direct synchronization of data, and the only datawide storage needed is the data FIFO/RAM, the depth of which is decided by the RTT of the mesochronous link. Since Starsync and the proposed architecture experience the same RTT and, hence, employ FIFO queues of the same depth, it means that Starsync will always pay the area/power cost of four more datawide registers. As shown in Fig. 6, for a one-cycle link delay, Starsync occupies 67% more area. Even though the gap remains constant over longer link delays, it is still quite high at up to 30%. The additional area overhead is considerable even for a five-cycle link delay.

V. CONCLUSION

Irrespective of the physical proximity of the sender and the receiver in a mesochronous clock interface, the proposed low-cost dual-clock FIFO combines mesochronous clock synchronization and buffering in a scalable manner. Data are safely transferred on the receiver side of a mesochronous interface without being explicitly synchronized. Synchronization involves only the single-bit push/pop flow-control signals. This implicit synchronization of data saves considerable amount of area/power, especially in the case of multicycle links, without introducing additional latency, or reducing throughput.

REFERENCES

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

[2] A. M. S. Abdelhadi and M. R. Greenstreet, "Interleaved architectures for high-throughput synthesizable synchronization FIFOs," in *Proc. 23rd IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2017, pp. 41–48.

[3] P. Teehan, M. Greenstreet, and G. Lemieux, "A survey and taxonomy of GALS design styles," *IEEE Des. Test Comput.*, vol. 24, no. 5, pp. 418–428, Sep./Oct. 2007.

[4] J. Ax, N. Kucza, M. Vohrmann, T. Jungeblut, M. Porrmann, and U. Rückert, "Comparing synchronous, mesochronous and asynchronous NoCs for GALS based MPSoCs," in *Proc. IEEE MCSoC*, Sep. 2017, pp. 45–51.

[5] W. J. Dally and J. W. Poulton, *Digital Systems Engineering*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[6] R. Ginosar, "Metastability and synchronizers: A tutorial," *IEEE Des. Test Comput.*, vol. 28, no. 5, pp. 23–35, Sep./Oct. 2011.

[7] R. W. Apperson, Z. Yu, M. J. Meeuwsen, T. Mohsenin, and B. M. Baas, "A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 10, pp. 1125–1134, Oct. 2007.

[8] D. Verbitsky, R. R. Dobkin, R. Ginosar, and S. Beer, "StarSync: An extendable standard-cell mesochronous synchronizer," *Integration*, vol. 47, no. 2, pp. 250–260, Mar. 2014.

[9] 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 *Proc. NOCS*, May 2009, pp. 244–249.

[10] D. Ludovici, A. Stranoy, G. N. Gaydadjievx, L. Beniniyy, and D. Bertozziy, "Design space exploration of a mesochronous link for cost-effective and flexible GALS NOCs," in *Proc. IEEE DATE*, Mar. 2010, pp. 679–684.

[11] S. Saponara, F. Vitullo, R. Locatelli, P. Teninge, M. Coppola, and L. Fanucci, "LIME: A low-latency and low-complexity on-chip mesochronous link with integrated flow control," in *Proc. 11th EUROMICRO Conf. Digit. Syst. Design Archit., Methods Tools*, Sep. 2008, pp. 32–35.

[12] A. Edman and C. Svensson, "Timing closure through a globally synchronous, timing partitioned design methodology," in *Proc. 41st Design Autom. Conf.*, Jul. 2004, pp. 71–74.

[13] M. Ghoneima, Y. Ismail, M. Khellah, and V. De, "Variation-tolerant and low-power source-synchronous multicycle on-chip interconnect scheme," *VLSI Des.*, vol. 2007, Mar. 2007, Art. no. 95402.

[14] I. Loi, F. Angiolini, and L. Benini, "Developing mesochronous synchronizers to enable 3D NoCs," in *Proc. Design, Autom. Test Eur.*, Mar. 2008, pp. 1414–1419.

[15] F. Vitullo *et al.*, "Low-complexity link microarchitecture for mesochronous communication in networks-on-chip," *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1196–1201, Sep. 2008.

[16] M. Paschou, A. Psarras, C. Nicopoulos, and G. Dimitrakopoulos, "CrossOver: Clock domain crossing under virtual-channel flow control," in *Proc. Design, Autom. Test Eur. (DATE)*, Mar. 2016, pp. 1183–1188.

[17] A. Psarras, M. Paschou, C. Nicopoulos, and G. Dimitrakopoulos, "A dual-clock multiple-queue shared buffer," *IEEE Trans. Comput.*, vol. 66, no. 99, pp. 1809–1815, Oct. 2017.

[18] G. Dimitrakopoulos, A. Psarras, and I. Seitanidis, *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective*. Cham, Switzerland: Springer, 2015.