StarSync: An Extendable Standard-cell Mesochronous Synchronizer

Dmitry Verbitsky[1,2], Rostislav (Reuven) Dobkin[2], Ran Ginosar[1], Salomon Beer[1]

[1]*EE Dept., Technion, Haifa 32000, Israel;* [2]*vSync Circuits LTD.*

Abstract —**StarSync, a mesochronous synchronizer, enables low latency and full throughput crossing of clock domain boundaries having same frequency but different phases. Full back pressure is supported, where the receiver can start and stop accepting words without any data loss. Variable depth buffering is provided, supporting a wide range of short and long range communications and accommodating multi-cycle wire delays. Burst data can also be accommodated thanks to buffering. Dynamic phase shifting due to varying voltage and temperature are mitigated by increasing the separation between write and read pointers. The synchronizer is exposed to metastability risk only during reset. It is suitable for implementation using standard cell design and requires neither delay lines nor other full custom circuits. It is shown that a minimum of four buffer stages are required, to mitigate skew in reset synchronization, in contrast with previous proposals for three stages.**

Keywords—**Synchronization, mesochronous, multi-synchronous, buffering, back-pressure**

1. Introduction

Modern technology generations lead to Systems on Chip (SoCs) integrating multiple IP modules placed on the same die. Clock distribution remains a major issue in such complex systems, because of the wire delay problem and because of delay variations. Distributing the global clock in a system with minimal clock skew is difficult due to the reverse scaling of global wire delay in nanoscale integrated circuits [1].

A fully asynchronous approach to global intra-chip communication would eliminate the clock distribution concerns and would make designs more modular since timing assumptions are explicit in the hand-shaking protocols [2]. Still, current design tools and IP libraries rely heavily instead on the synchronous paradigm, making intermediate solutions more attractive and affordable in the short run. Generic solutions assuming no knowledge of clock relations may suffer from inferior throughput or may require custom circuits approach [3][4]. A trade-off between synchronous and asynchronous approaches consists of the mesochronous scheme [5][6] or the multi-synchronous method [7][8].

In a mesochronous system, a single clock signal is distributed to the various modules in the design with an arbitrary amount of space-dependent time-invariant phase offset (i.e., clock skew). Mesochronous synchronization enables architecture scalability and may also mitigate the skew constraints in the clock tree synthesis process, resulting in higher clock rate, lower power and faster back-end turnarounds [9].

The contributions offered by this paper include an extensive survey of previous work on mesochronous and related synchronizers, a proof that at least four buffer stages are generally required in a mesochronous synchronizer based on a cyclic buffer (due to initialization issues), an analysis of the impact of wire delays and clock drifts on mesochronous synchronization, and a description of StarSync, a complete practical full throughput synchronizer that can be implemented using standard cells on ASIC and FPGA, that supports back pressure, provides extensible buffering, enables both short and long interconnects, accommodates drifting or multi-synchronous clocks and achieves high MTBF.

The paper is organized as follows. Sect.1 discusses related work. Sect.1 describes StarSync architecture and design considerations. In Sect.1, StarSync performance is analyzed, and StarSync is compared with a standard two-clock FIFO. Conclusions are drawn in Sect.1.

2. Related Work

A large body of previous research related to mesochronous and similar synchronizers is reviewed in this section and summarized in Table 1.

In practice, often the standard two-clock FIFO synchronizer [10], which can bridge any two clock domains and not merely mesochronous ones, is employed even when synchronizing mesochronous domains, primarily thanks to its extensive heritage in actual systems, its ready availability and its robustness, in spite of its latency disadvantage (as discussed in Sect. 4.5).

A common approach to the design of mesochronous synchronizers consists of delaying either data or the clock signal to sample data reliably when they are guaranteed to be stable. Fig.1 shows a typical scheme of delay-line based mesochronous synchronizer [5]. Signal *X*may change close in time to the sampling clock *clk*, leading to metastability of the sampling XS flip-flop. By changing the delay line settings, the relation between data and the clock is modified so that data transitions happen outside the 'keep-out' region determined by the two delay lines t_d and $t_{cy}-t_d$, where t_{cy} is the clock period and t_d is a half-width of the keep-out region, which must be accurately matched to provide minimal latency and at the same time meet the setup and hold requirements of *XS* flip-flop. A similar approach requiring only t_d delay line is suggested in [7][8][11].

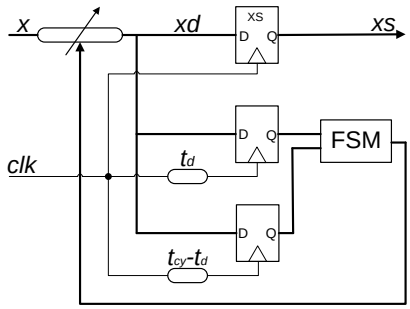


Fig.1. Delay-line synchronizer([5], *figure 10-9*)

A digitally calibrated delay line is employed for shifting the clock of the write-side FSM in [12], while the rest of the transmitter clock domain is clocked by a non-shifted version of the clock. Unfortunately, the principles of choosing the delay value as well as the structure of the synchronizer circuit employed for pointer synchronization were not disclosed. In [13] a delay line is employed for setting the relative skew between the write and read clocks, and the delay value is determined at design time by static timing analysis (STA). The authors employ double data rate source synchronous communication and sample data at the receiver side using a dual stage buffer. Ref. [14] generalized the method of [13] to construct a token-based FIFO-like transfer link. In [15] a delay-line on the read clock was employed in order to safely sample encoded write pointer at the receiver side of the proposed FIFO. A self-timed single stage FIFO for mesochronous communication was proposed by [16]. Its implementation is quite simple, but it requires delay lines for proper clocks adjustment.

A different approach for dealing with an unknown clock phase shift involves a phase detector circuit (Fig.2, following [19]). In [17] a phase detector is used to predict conflicts and to delay the write clock. Similar methods of phase detection for conflict prediction are proposed in [18] and [19], where for each predicted conflict the write clock is shifted as well. Conflict detectors to track drifting phase shifts between multi-synchronous clock domains are employed in [8]. In [20] and [11] conflict detectors are employed to prevent metastability when using periodic clocks.

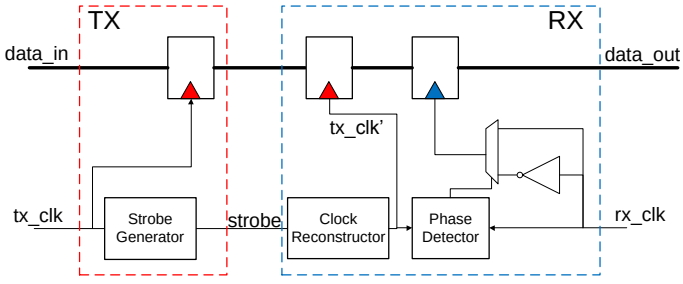


Fig.2. Phase Detection Synchronizer [19]

An all-digital synchronizer that exploits the predictability of periodic clocks is proposed by [21]. Its advantages include low latency, handling various clock frequency relations and robustness to clock jitter and phase drift. However, it incurs relatively high implementation complexity, requiring the tuning of several parameters to guarantee safe synchronization. It also uses custom design delay lines.

Ref. [22] uses a learning-phase-based interface for mesochronous synchronization. The transmitter generates a strobe signal, the delayed version of which is sampled at the receiver by both a positive and a negative edge-triggered flip-flops. Before being analyzed, samples are synchronized with N-flop synchronizers in the receiver clock domain. After the learning phase, the receiver knows on which clock edge it should sample data, by detecting the first sample of delayed strobe that is different from the previous one. The advantages of that approach include the ease of implementation, capability to cope with various non-idealities (like clock and data jitter), low latency and reduced overhead. On the other hand, the design uses delay lines and offers no flow control.

Long wire delays between communicating modules typically incur additional skew. An *N*-depth mesochronous synchronizer is proposed in [23] to deal with global wire delays, estimated using STA and used to define the spread of the write and read pointers of the *N*-depth synchronizer. Ref. [24] suggests a similar technique for long wires, dynamically calculating the pointer spread, using a phase detector. Mixed timing relay stations were introduced in [25] for handling long wire interconnects, and for interfacing synchronous and asynchronous domains.

Delay-line based synchronizers are mostly suitable for full custom designs. They require clock tree modifications and multiple instantiations for multi-bit applications. For SoCs which are based on standard cell design, including both ASIC and FPGAs, and which do not permit any fine-tune clock tree modifications, another approach to mesochronous synchronization is required. The principle structure of such a synchronizer is shown in Fig.3[5]. This mesochronous synchronizer employs cyclic write and read pointers, *wp* and *rp* respectively, with a certain initial spread to allow collision-free write and read operations. The write pointer is synchronous to *wclk* and the read pointer is synchronous to *rclk*, while the clocks are mesochronous. References [26] and [27] employ a similar structure; the goal is to provide time for multi-stage metastability resolution, avoiding a serial synchronization chain.

Ref. [28] shows a constraint-based forward-path synchronizer design without back-pressure. Ref. [29] employs the three-element cyclic buffer mesochronous synchronizer as shown in Fig.4. The architecture comprises two synchronization paths. The ‘data sync’ path synchronizes data that are sent from the transmitter to the receiver. The ‘ctrl sync’ path synchronizes back-pressure signaling in the reverse direction. Both synchronization paths employ cyclic buffers, each consisting of three stages. Buffers are managed by cyclic counters, and their initial spread should avoid metastability. Counters of the ‘ctrl sync’ run continuously, whereas counters of the ‘data sync’ are stopped each time the *Stall* signal is received (TX/RX Enable in Fig.4).The circuit is tolerant only to small phase drifts; larger phase drifts may result in unstable data being sent to the output of the synchronizer.

Data bursts can be supported in synchronizers that implement

back-pressure. Another mechanism that enables data bursts was described in [30].

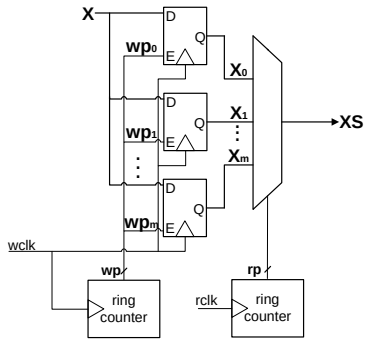


Fig.3. m-elements cyclic buffer mesochronous synchronizer [[5], figure 10-13]

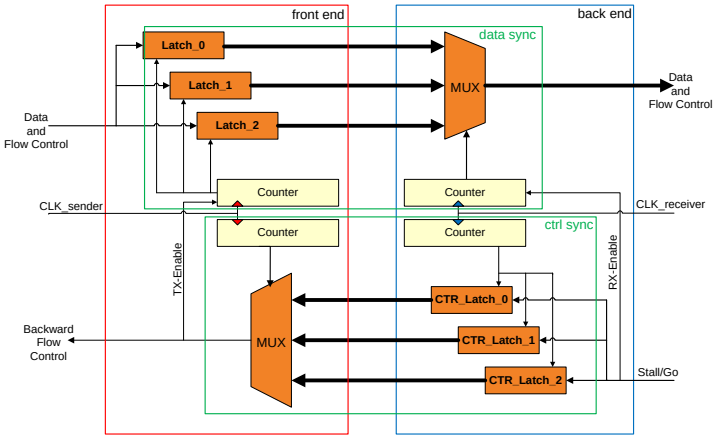


Fig.4.A Three-Element FIFO Synchronizer with Back-Pressure [29]

StarSync is compared to previous work in the rightmost column of Table 1. More specifically, the contributions made in this paper include:

1. The proof that at least four buffer stages are generally required in a mesochronous synchronizer based on a cyclic buffer (due to initialization).
2. Analysis of the impact of wire delays and clock drifts.
3. A complete practical full throughput synchronizer that can be implemented using standard cells on ASIC and FPGA, supports back pressure, provides extensible buffering, enables both short and long interconnects, accommodates drifting or multi-synchronous clocks and achieves high MTBF.

StarSync enables data transfer between mesochronous clock domains. The receiver clock is never stopped or modified, and the write and read pointers are never stopped as well. The architecture supports full backpressure and buffering for data burst transfers and is also suitable for long range communications. In addition, relative to other implementations, in StarSync only a single cyclic-counter is employed at each side of the synchronizer. The initial spread of the pointers and the number of stages in StarSync are key design considerations, and are discussed with respect to different operating conditions. We show that three synchronization stages are insufficient to assure correct initialization and may lead to metastability; rather, a minimum of four stages is required in general.

Table 1 : Previous work on mesochronous and related synchronizers

Category	Previous Work	This Paper: StarSync
Synchronizer type	[5][7][8][11][12][13][14][16][17][18][19][22][23][24][26][28][29]: Data synchronizer [15][21][25]: Control synchronizer	Data synchronizer
Delay Line	[5](figure 10-9)[7][8][18][21][22]: Data delay line [12][15][16]: Clock delay line [13][14]: Static clock delay line	No delay line
2-clock FIFO	[5](figure 10-9)[12]: Cyclic buffer [14][15]: Bi-synchronous FIFO [23][24][28]: Cyclic buffer (STA based read-write pointer spread) [25]: mixed-clock FIFO [26]: Cyclic buffer with fixed pointer spread [29]: Cyclic buffer of 3 stages	2-clock FIFO based on 4 stage cyclic buffer
Multi-Phase Sampling	[7][8][11][20][22][27]	No multi-phase sampling
Phase Detector	[17][18]: phase detector of two phases per cycle [5][7][11][20][21][22]: Phase detector with delay line [24]: reset-only phase detector comparing normal and delayed data	No phase detector
Back pressure	[14][15][11][21]	Supports back pressure
Support long interconnect	[23][24][28]	Supports long interconnect with back pressure
Suitable for standard cells and FPGA	[5](figure 10-13)[23][25][28][29]	Suitable for standard cells and FPGA
Support drift or multi-synchronous clocks or periodic clocks	[5](figure 10-13)[29]: Drift tolerance up to 1 cycle [16]: Drift tolerance up to 2 cycles [7][8][20]: Multi-synchronous clocks with re-synchronization and training [11]: Periodic clocks [22]: mesochronous and plesiochronous clocks	Support multi-synchronous clocks without re-synchronization
Latency	[7][8][11][16]: Latency 1 or less cycles [22]: Latency between 1 and 1.5 cycles [5][12][15]: Latency between 1 and 2 cycles [17][21]: Latency between 1.5 and 2 cycles [13][14]: Latency 2 cycles [18][19]: Latency between 2 and 3 cycles [7]: Latency equals phase difference [23]: Latency equals pointer spread [24]: Latency at least interconnect delay divided by clock cycle [25]: Latency between 3 and 3.5 cycles	Latency between 1 and 3 cycles, plus interconnect delay
Throughput	[25]: Full throughput or less All others: Full throughput (one word each clock cycle)	Full throughput

3. StarSync Architecture and Extensions

3.1. Conceptual description

The synchronizer connects a sender to a receiver and consists of a TX and a RX blocks. The TX block uses the same clock as the sender, and RX uses the receiver clock. The sender can send data via the synchronizer to the receiver, and the receiver may apply backpressure to the sender, asking to stop sending data. Ideally, the sender should be able to send an infinite stream of data (one new word each cycle), but the receiver can use backpressure to stop data transmission when it cannot

receive more data. Still, the sender should be allowed to support a burst mode by sending a continuous burst of the pre-defined number of data words (*L*) without interruption. Note that it takes a certain delay from the time of issue of backpressure by the receiver and until the sender is notified of that backpressure. Meanwhile, some data words have already accumulated inside the synchronizer, and the sender continues to send data until it receives the notice of backpressure. We do not wish to lose these data, and a FIFO is added at the RX block to keep that data until the receiver agrees to receive it.

Three StarSync extensions are described. First, the FIFO at the RX block may be extended, to enable receiving a burst of *L* words even if the receiver has applied backpressure. The actual backpressure signal is delayed to the sender until after the sender has sent all *L* words of the burst. StarSync can support long interconnecting wires between TX and RX. Such wires may incur multi-cycle delay between TX and RX. To support that, the RX FIFO can be further extended, and the synchronization buffers in TX and RX can also be further extended.

Last StarSync also supports correct multi-synchronous operation in the presence of cumulative dynamic phase shift between the two mesochronous clocks. That shift can accumulate beyond one clock cycle. To allow that, longer synchronization buffers are employed on both TX and RX.

3.2. Architecture

StarSync architecture is shown in Fig.5. The mesochronous synchronizer employs two cyclic buffers (TX_BUF and RX_BUF), each consisting of *four* parallel stages (as explained in Sect. 0 below). TX_BUF is a data buffer, responsible for data transfer from the transmitter interface to the receiver. TX_BUF holds both forward tokens and data. Forward data valid (DV) tokens indicate the data validity. RX_BUF holds backward tokens, which indicate the amount of free space at the receiver side.

Data Transfer

The data and the forward DV tokens (PUSH signal converted into data valid DV signal) are written cyclically into DATA_REG_*x* and DV_FF_*x* registers with TX_CLK. The cyclic WR_COUNTER is never stopped. The same is true for the RD_COUNTER, which is clocked by the mesochronous RX_CLK. Thanks to the initial spread (discussed below in Sect.0), metastability is avoided when reading data from TX_BUF using the read pointer RP. The read data are output over the RX_DATA and EMPTY lines. The EMPTY signal is effectively an inverted copy of the DV signal.

Backpressure

The receiver side can stop the transfer by de-asserting the POP signal, which immediately causes the data from TX_BUF to be saved into the Single-Clock FIFO. Note that if there are valid data in TX_BUF or in FIFO then the EMPTY signal remains low, since it always indicates the presence of valid data regardless of whether the data are held in TX_BUF or in FIFO.

The backward token in Token_FF_*x* is either 'ClearToSend' ('1'), allowing writing into DATA_REG_*x* and DV_FF_*x*, or 'DoNotSend' ('0'). In a basic (minimal buffering) configuration, the FIFO depth is equal to the depth of TX_BUF (four buffers), to allow clearing of TX_BUF into FIFO, since the pointer counter is never stopped. 'DoNotSend' tokens are written into RX_BUF when POP is de-asserted. When Token_FF_*x* is 'DoNotSend', writing data into the corresponding DATA_REG_*x* and DV_FF_*x* is blocked, since the data register is occupied. FULL signal is asserted, providing back-pressure. Data transfer is resumed when POP signal is re-asserted. After asserting POP, data from FIFO are output on RD_DATA. In parallel, RX_BUF is filled with 'ClearToSend' tokens that are passed to the transmitter. Once a 'ClearToSend' token is selected by WP, FULL is de-asserted, enabling a write operation. During subsequent write and read operations, the data at the receiver side are pushed into the FIFO as long as it is not empty. Once the FIFO becomes empty (e.g., thanks to idle cycles at the transmitter side) the data bypass the FIFO and are output with normal latency.

Token Management

The token buffer RX_BUF is managed in the following way:

- All token buffer stages are initialized to '1' ('ClearToSend')
- For valid read cycle (namely, POP is high) the following operations are performed:
  - Valid Token ('1') is written into the current RX_BUF stage (pointed at by read pointer RP), indicating that the corresponding TX_BUF stage is empty and is ready to receive new data.
  - When FIFO is not empty, current data from TX_BUF is written into the FIFO (only if the current TX_BUF register holds valid data, namely the corresponding DV is high) and the data for RD_DATA bus is popped out of the FIFO. Otherwise (when FIFO is empty) valid data from TX_BUF are conveyed directly to the output (FIFO is bypassed).
- For stall cycle (namely, POP is low) the following operations are performed:
  - Valid data from TX_BUF are pushed into the FIFO.
  - When THRESHOLD is low (namely, there is still room in the FIFO), a Valid token is written into RX_BUF.
  - When THRESHOLD is high (namely, the FIFO has room only for words that are already in TX_BUF but no room for additional new words), Invalid token ('0') is written to RX_BUF. Note that the FIFO has room also for new words that may be inserted into TX_BUF until the Invalid backward token has propagated and toggled the FULL signal.

Since TX_CLK and RX_CLK have the same frequency, but a constant relative skew, the initial values of the write and read pointers should be carefully set to eliminate contention (WAR or RAW hazards). The initial write pointer is set to 0, and the initial read pointer is set to 2 (as detailed below in Sect. 0).The initial separation is preserved throughout subsequent operation thanks to the mesochronous clock relationship.

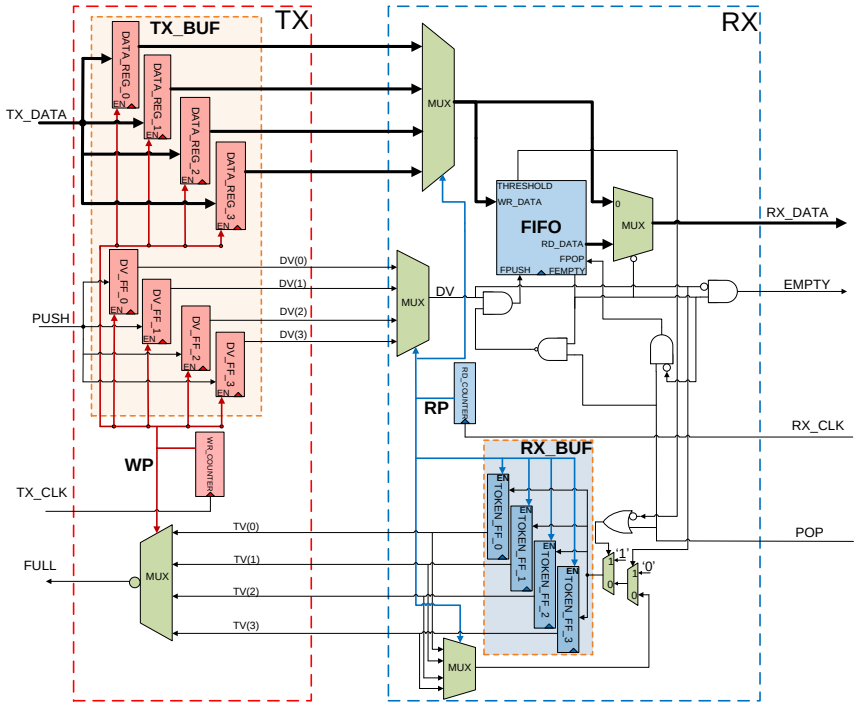


Fig.5. StarSync Architecture

At the output, the RX_DATA multiplexer chooses the output of the FIFO, when it is not empty, or TX_BUF otherwise. Additional multiplexing selects EMPTY, which is '0' if either internal FIFO is not empty or internal FIFO is empty but the TX_BUF holds valid data, and '1' otherwise.

The basic StarSync architecture described above can be extended to support the bursts, long interconnect delay and multi-synchronous clocking, as follows.

3.2.1 Support of Data Burst and Buffering

As described above, the minimal FIFO depth equals the depth of TX_BUF (depth of four in the basic configuration). The minimal depth is dictated by the maximal number of words inside TX_BUF. Since the read out of TX_BUF is never stopped, the words are saved aside in the FIFO when POP is de-asserted.

Burst transfer is supported by making FIFO deeper. For a data burst of length $L>4$ the FIFO depth is set to L. The THRESHOLD signal is asserted when the number of data words in FIFO reaches the predefined threshold of $L-4$. As described below, both POP and THRESHOLD affect token generation for RX_BUF.

3.2.2 Support of Long Wires

Two different modes of synchronizing over long interconnect are described. The interconnect is considered long when the wire delay is significant, e.g., data takes $m>0$ clock cycles to travel over the wire:

1. Source synchronous communication, where the entire synchronizer is placed at the sink of the communication link (Fig.6a) and the data transmitted on the link are accompanied by a STROBE. A similar approach was presented in [24]. In that case, the long lines are not part of the synchronizer, and may be implemented as either wave-pipelined links or registers may be inserted over the long wires.
2. Split synchronizer: Long wires between the TX and RX sides of the synchronizer (Fig.6b).

Two variants of the synchronizer are employed for source synchronous communication (Fig.6a). If no backpressure is needed, the basic StarSync of Fig.5 may be used. Otherwise, FIFO should be extended to depth $4+m$. This is required because, when the receiver RX is stopped, the FULL indication may arrive at the source module after a few clock cycles due to the long interconnect delay. By increasing the depth of the FIFO, the synchronizer deals with this case without any latency overhead. The number m of additional required FIFO stages can be computed using Static Timing Analysis (STA). Note that any spare stages added for safety affect only the area; neither throughput nor latency are affected.

In the split synchronizer case (Fig.6b), in which the TX and RX parts are placed far away from each other, long delay is incurred both over the TX-to-RX data path and over the RX-to-TX token path. When no backpressure is needed, only the TX-to-RX interconnect delay is taken into account. In this case, the synchronizer should assure that the data are read before the same buffer is written. Since it takes time for the data and for the DV signal to reach RX, the synchronizer TX_BUF needs to be extended to cover for the longer interconnect delay. While in a short wires synchronizer four TX_BUF stages suffice (Fig.7a), for an interconnect delay of m cycles, additional m stages are added (Fig.7b), resulting in a $4+m$ stage buffer. The initial spread between the pointers is two in both cases, and *RP* points to a TX_BUF entry only after the entry becomes valid, finishing its propagation through the

interconnect. Red color was used to define the cycles where it is not allowed to sample DATA_REG_0, because it is not yet ready. Yellow was used to define the cycle where sampling of DATA_REG_0 is not safe due to initialization case where write and read pointers spread is one. And finally, green was used to define the cycle where DATA_REG_0 sampling is safe. The required additional number of stages *m* is computed using STA. When backpressure is desired, the additional latency of the backpressure signal TV (Token Valid) should be taken into account, adding in a similar way additional 2*m* stages to both TX_BUF and RX_BUF (Fig.7c). The effective rate of each line is 1/2m, because new data can be sent on the same line only once per 2m cycles, whereas *m* is chosen to be the maximum delay over all control and data lines. In that way, when DV and data lines are sampled, they are all stable.
In addition, similarly to the source synchronous case, FIFO depth is extended to 4+2*m* in order to compensate for that delay. This split synchronizer is useful for mesochronous NoC, where TX can serve as the output ports of a NoC router and RX is the input port of the next router [32].

3.2.3 Support of Multi-Synchronous Communications

Mesochronous clocks may suffer from phase drifting during operation, due to temporal and spatial variations in voltage and temperature. In this work we analyze dynamic phase shift in the range of [0, ±T) between the clocks, where T is the clock period. We show in the next section that six synchronization stages can deal with such phase drifts.
In general, each cumulative T-long phase shift requires adding two synchronization stages, to keep the pointer spread symmetric in both directions, coping with both positive and negative clock phase shifts. Each stage incurs an additional DATA_REG, DV_FF and TOKEN_FF registers (see Fig.5). It means that in order to cope with phase shift of [0, ±*k*·T) we need 4 + 2*k* stages in each of TX_BUF and RX_BUF.
Note that the extensions described in the ensuing three sub-sections, related to bursts, to long wires and to multi-synchronous communications, are independent of each other and may be applied in any combination. Note further that all three extensions lead to requirements regarding the FIFO depth L, stemming from either burst length, or from line length as determined through STA, or from drift requirements, or any combination thereof; these requirements result in a feasible FIFO depth of L words, and correct operation in all cases (avoidance of FIFO overflow) is assured by means of back-pressure.

3.3. Initialization

An asynchronous reset is employed for circuit initialization. To properly reset multiple clock domains, the reset signal must be synchronized with each one of their respective clocks.
Fig.8 shows a typical case of two two-flip-flop reset synchronizers for two clock domains (more than two stages may be required to achieve a desired MTBF). These synchronizations may lead to non-deterministic timing of the release of the multiple synchronized reset signals: due to possible metastability, each synchronizer may produce the trailing edge of reset after either *k* or *k*+1 cycles of its respective clock, for some *k*.

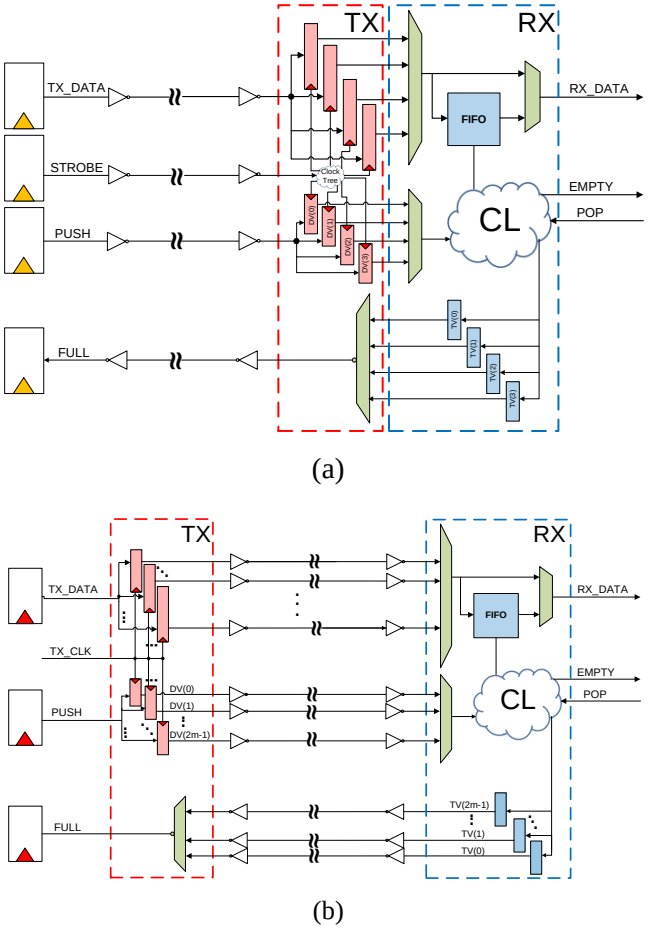


Fig.6. Communication over m-cycles long interconnect

(a) Source synchronous communication
(b) Split synchronizer: Long wires between TX and RX (DV and TV signals)

For each clock domain the reset is asserted asynchronously and instantaneously, but is released synchronously, leading to skew of [0, ±T), where T is the clock period. The waveforms in Fig.9 describe the operation for two possible maximum skew scenarios that can occur: either one of the synchronized release of the reset signals precedes the other by a skew no more than T.
Notice that when the wire delays of global 'RESET' signal to WR and RD reset synchronizers are different ($\Delta T1$ and $\Delta T2$ in Fig.8), an additional skew may be incurred. We assume that this skew can be mitigated by reset tree balancing toward R1 and G1 reset synchronizers, since this is a relatively small tree that can be lumped within a small area on the chip, and the balancing can be performed by conventional routing tools. Therefore, skew on the RESET wires towards the synchronizers is disregarded in the following analysis. When resets are asserted, write and read pointers are set to their initial values with a spread of at least 2 (0 and 2 respectively, for short interconnect, or 0 and *k* for split synchronizer with backpressure).

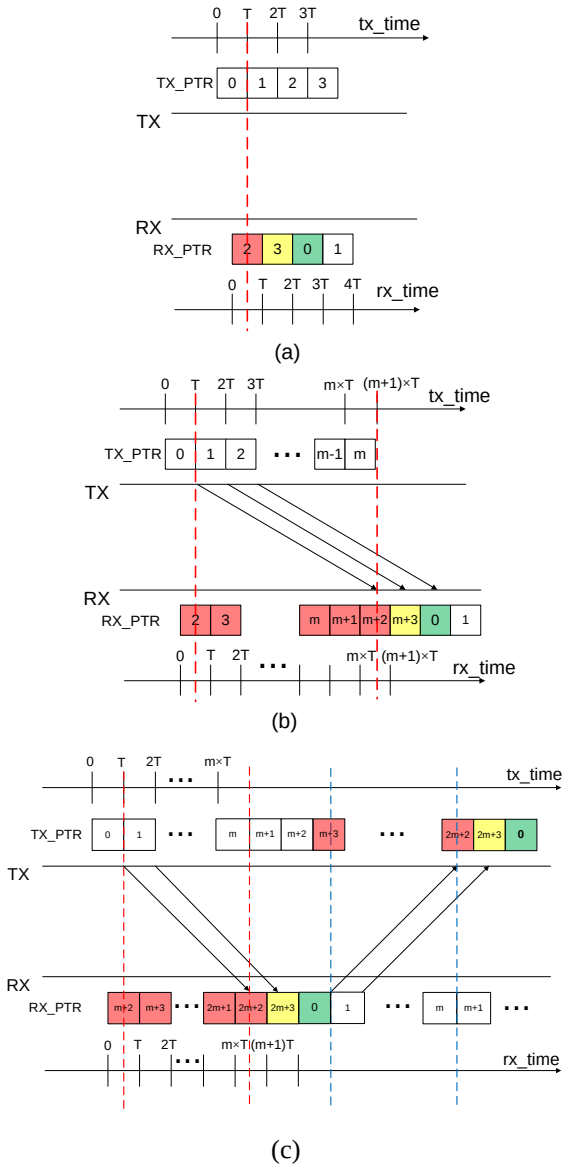


Fig.7. data/token transitions (a) short wires (b) long wires split synchronizer, no backpressure
(c) long wires split synchronizer with backpressure

These initial values are explained as follows, showing that a pointer spread of less than two is insufficient. First, consider the case of zero initial pointer spread as in Fig.10. In this example, both reset synchronizers are resolved after the same number of cycles. In addition, clock phase between the clocks is less than the setup-hold window width of RD_CLK (green area in Fig.10). The problem is that DATA_REG_i and DV_FF_i (see Fig.5) change in the metastability region of RD_CLK violating its setup time.

Next, consider initial pointer spread of one as in Fig.11. In this case, if one synchronizer takes longer to resolve than the other, one pointer gets incremented in the first cycle while the other pointer does not, they become equal and we face again the risk of concurrent read and write. So only the initial spread of two (or more) can guarantee continuous maintenance of minimal spread of one, because reset synchronization cannot reduce the

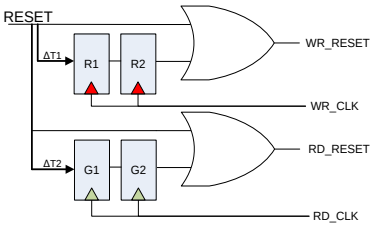
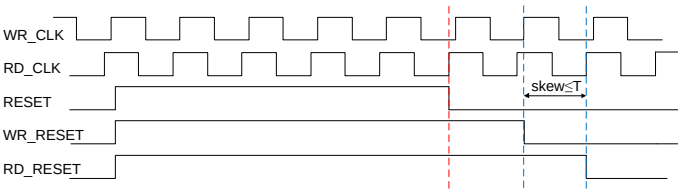
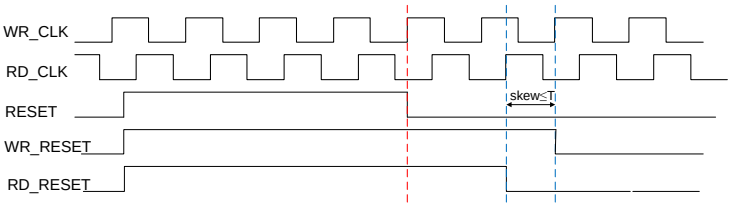


Fig.8. Synchronization of asynchronous RESET in two different clock domains



(a) WR_RESET precedes RD_RESET

(rising RD_CLK just misses falling RESET)



(b) RD_RESET precedes WR_RESET

(rising WR_CLK just misses falling RESET)

Fig.9. Reset: two maximum skew scenarios (a) positive skew up to T (b) negative skew up to T

spread by more than one. The same is true in the opposite direction when tokens are written to TOKEN_FF_i in RD_CLK domain and are read in WR_CLK domain. In order to maintain a minimal symmetric pointers spread of two, the minimal cyclic buffer depth should be equal to four, while the buffer depth of three can provide a spread of two in only one direction.

The synchronizer can be enhanced to support multi-synchronous domains subject to a dynamic phase drift. Here we analyze a phase drift limited to a single clock cycle. In case of a phase drift of a whole cycle the spread between the pointers is reduced and the situation described in Fig.11 is again possible. In order to avoid this, additional spread of one place must be added between the pointers. That means that the initialization setup would be six register stages with initial write pointer = 0 and initial read pointer = 3. Note that the pointer spread must be symmetric in both directions to deal with both positive and negative clock phase shifts. Thus, each additional phase shift of T adds two synchronization stages, leading to $4+2k$ stages for phase shift of $[0, \pm kT)$.

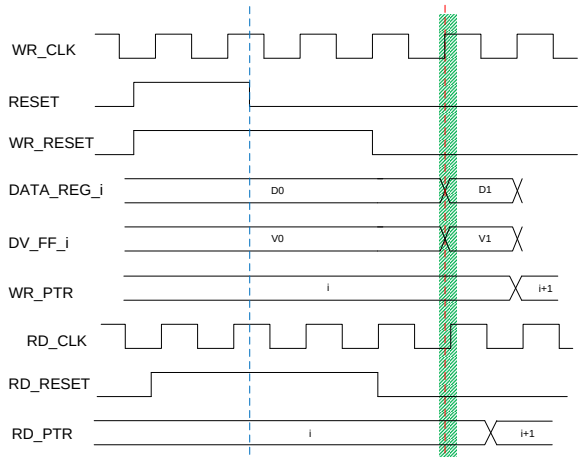


Fig.10. Setup time violation when pointers are initialized to the same value

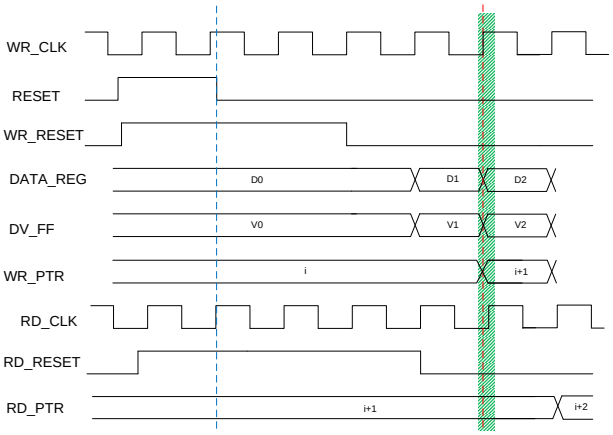


Fig.11. Setup time violation when pointers are initialized with spread of one

4. PERFORMANCE

In this section, StarSync performance is analyzed. Forward latency, throughput, area and power requirements are discussed, and StarSync is compared to a two-clock FIFO.

4.1. Forward Latency

The forward latency is defined as the time from the data being placed at the synchronizer input until it appears at the synchronizer output. The forward latency varies in the range (T, 3T), depending on the initial pointer spread and on the relative phase difference of the two clocks. Fig.12 and Fig.13 show the forward data latency when RD_CLK has a positive or negative phase shift, respectively, where Φ is the absolute value of the phase shift.

4.2. Throughput

StarSync provides maximal throughput, namely a data word can be transferred on each clock cycle, when the receiver side is ready and if there is no backpressure. When backpressure is applied, the throughput is linearly affected as shown in Fig.14.

4.3. Area

StarSync area consists mostly of registers and flip-flops.

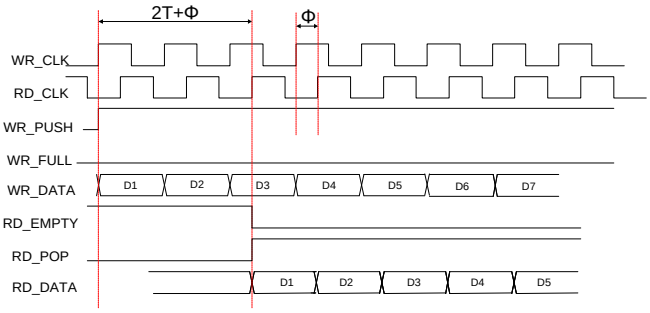


Fig.12. StarSync forward latency 2T+Φ (WR_CLK precedes RD_CLK)

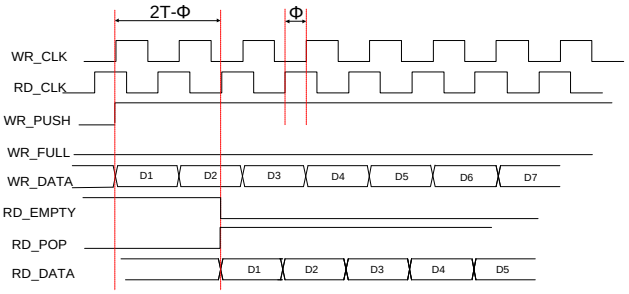


Fig.13. StarSync forward latency 2T-Φ (RD_CLK precedes WR_CLK)

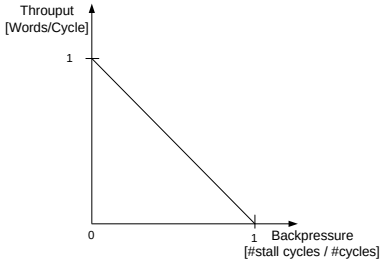


Fig.14. Throughput vs. RX backpressure

The number of registers depends on the data word width, on the depth of the synchronization buffers (BufDepth), and on the FIFO depth (FifoDepth), as shown in Eq.(1). Note that FifoDepth=max(BufDepth, L), where L is the data burst requirement. The register data count comes from two main sources: one is the TX and RX BUFs and their pointers WP and RP, and the FIFO. *BufDepth* should be greater or equal to four. Increasing *BufDepth* (in the cases of split synchronizer for long interconnect and multi-synchronous domains) has the most significant impact among all parameters, as it affects also the value of *FifoDepth*.

$$\begin{aligned} Area \propto & \\ & BufDepth \cdot (DataWidth + 2) + && \#\text{TX\&RX BUFs} \\ & 2 \cdot \lceil \log_2(BufDepth) \rceil + && \#\text{WP \& RP} \\ & FifoDepth \cdot DataWidth + \lceil \log_2(FifoDepth) \rceil && \#\text{FIFO: data and control} \end{aligned} \quad (1)$$

4.4. Dynamic Power

StarSync implementation is based only on standard library cells and therefore, StarSync power can be assessed using conventional digital design power analysis tools that rely on

the power values specified for the library. Since power analysis is straightforward, in this section we shortly review the impact of StarSync configuration parameters on the power. For the long-wires extension (Sect. 3.2.2), additional power is incurred, required for driving the long interconnect wires.

A single register inside each buffer is enabled per cycle thanks to the cyclic enable pointers. Thus, data toggles in only about one fourth of the circuit. In addition, FIFO works only when backpressure is applied, while during continuous data transfer it can be clock-gated. Therefore, for a continuous data transfer, the power is proportional to about one fourth of the TX/RX_BUF registers. Clock gating can be employed in all the disabled registers, controlled by WP and RP. Thanks to the mesochronous clock relation (and the fact that typically the two clock phases are different from each other), peak power is reduced since TX and RX clocks do not toggle simultaneously.

$$Power \propto \frac{1}{4}\times\left[BufDepth\cdot(DataWidth+2)+2\cdot\lceil\log_2(BufDepth)\rceil\right] \quad (2)$$

4.5. Comparison to two-clock FIFO

The structure of a standard two-clock FIFO [10] is shown in Fig.15. Incoming data are pushed into the FIFO when write enable is asserted (push). On the subsequent cycle, a gray coded write pointer wp_g is sent to the synchronizer, which usually consists of at least two stages. The synchronized value of the write pointer wp_sync is then compared to the current value of the read pointer to assess the amount of unread data inside the FIFO, setting the empty signal accordingly. When the empty is de-asserted, pop signal can be applied (for fast response the pop signal can be asserted immediately after empty de-assertion, on the same clock cycle). In a standard FIFO protocol, the data appear at the FIFO output on the next clock cycle. For fall-through FIFO the data appear along with the empty de-assertion.

Fig.16 shows the waveform diagram of the two-clock FIFO operation and the forward latency for the standard FIFO protocol. The total worst case latency is four clock cycles. For the fall-through FIFO it can be reduced to three.

In terms of throughput, both structures provide the maximal throughput of one word per cycle (WPC).

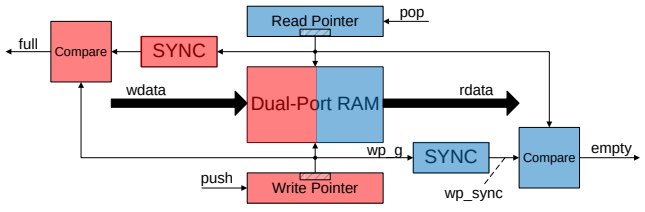


Fig.15. Two-clock gray fifo structure [10]

The area expression for the two-clock FIFO is shown in Eq.(3). We assume here that the two synchronization stages are enough to obtain high reliability, otherwise the factor of 4 in Eq.(3) should be increased.

StarSync is compared to two-clock FIFO in Table 2. For StarSync, only the flip-flops of the reset synchronizer may become metastable.

$$Area = \begin{matrix} 2\cdot\lceil\log_2(BurstLength)\rceil+ & \#\text{WP \& RP} \\ 4\cdot\lceil\log_2(BurstLength)\rceil+ & \#\text{SYNCs} \\ BurstLength\cdot DataWidth & \#\text{Buffers} \end{matrix} \quad (3)$$

Thus, the MTBF of StarSync equals the MTBF of the reset synchronizer. Using regular MTBF expression [30], we can compute this MTBF, as in the following example. Assume sampling clock frequency F_C=100MHz, reset toggling frequency F_D=1 toggle/day, settling time T=10 ns, τ=40 ps for 90nm technology, and W=10 ps:

$$\text{MTBF}=\frac{e^{\frac{S}{\tau}}}{W\cdot F_C\cdot F_D}=\frac{e^{200}}{10^{-11}\cdot 10^{8}}=e^{200}\cdot 10^{3}\,days$$

Evidently, this value of MTBF is practically unlimited.

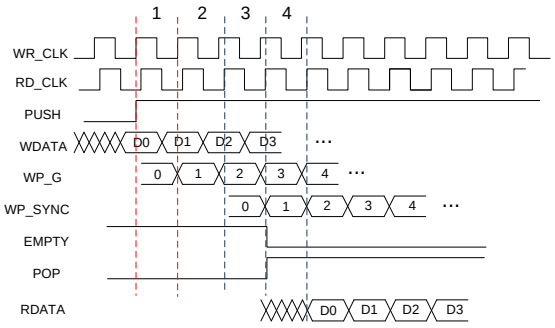


Fig.16. Two-Clock FIFO latency

StarSync latency is at most the same as the minimal latency of the two-clock FIFO, while the same maximal possible throughput is provided. The latency of the two-clock FIFO depends on the clock rate and the technology as for certain cases a longer delay is required to resolve metastability inside SYNC modules. StarSync, however, does not incur such overhead and always incurs three or fewer cycles forward latency (down to one cycle), with an unlimited MTBF.

In addition, the two-clock FIFO does not support long wire communication (Sect.3.2) since it cannot be distributed.

Table 2 : Comparison of StarSync and two-clock FIFO

Parameter	StarSync	Two-Clock FIFO (with two stage gray-code synchronizers)
Latency (cycles)	1−3	3−4
Throughput (WPC)	1	1
MTBF	Practically Unlimited	Technology & Operating conditions dependent

To provide full-throughput while accommodating backpressure and supporting bursts, the two-clock FIFO is usually designed with a depth of at least eight places (although five stages suffice [33]). For example, the minimal depth of Xilinx two-clock FIFO is 16 [34], while the minimal depth of Altera two-clock FIFO is 8 [35]. While the two-clock FIFO can support data bursts by increasing its depth, it cannot support long wires, unlike StarSync. On the other hand, a two-

clock FIFO can support unrelated clock frequencies while StarSync is optimized for mesochronous and multi-synchronous clock domains. The critical path of StarSync resides inside the single-clock FIFO; thus, the maximum allowed frequency of operation can be optimized in the same manner as in any synchronous, single-clock circuit, and does not depend on the other parts of StarSync. Regarding area comparison, StarSync occupies about the same area as the full throughput two-clock FIFO. When the data width is held constant, both Eq. (1) and Eq. (3) converge to the same value of BurstLength×DataWidth. In terms of power, we note that the two-clock FIFO must always work with its internal dual-port RAM, whereas StarSync may bypass the RAM consuming less dynamic power.

5. Conclusions

In this paper data transfer between mesochronous clock domains was discussed. StarSync, a new mesochronous synchronizer with back-pressure and burst support was presented. The paper shows that a minimal depth of four buffers is required to avoid timing violations, when a real reset circuitry is considered. This is in contrast with previous publications that employed three buffers. StarSync provides low latency and full throughput, while requiring less area than a standard two-clock FIFO. StarSync can be configured to support both long range source-synchronous communication and long wires between the transmitter and receiver sides of the synchronizer. In addition, StarSync may be extended to support dynamic clock phase drift and custom burst length. StarSync relies only on synthesizable logic and standard cell libraries and therefore it is suitable for any SoC, FPGA and NoC applications.

References

[1] E.G. Friedman, Clock Distribution Networks in Synchronous Digital Integrated Circuits, In Proc. of the IEEE 89(5),pp. 665-692, 2001.

[2] J. Sparso and S. Furber, Principles of asynchronous circuit design - A systems perspective, Kluwer Academic Publishers, 2001.

[3] R. Dobkin and R. Ginosar, Fast Universal Synchronizer, Proc. 18th International Workshop PATMOS, Lisbon, Portugal, pp.199-208, 2008.

[4] R. Dobkin and R. Ginosar, Two Phase Synchronization with Sub-cycle Latency, Integration, the VLSI Journal, 42(3) , pp.367-375, 2009.

[5] W.J. Dally and J.W. Poulton, Digital Systems Engineering, Cambridge University Press, 1998.

[6] T.H. Meng, Synchronization Design of Digital Systems, Kluwer Academic Publishers, 1990.

[7] R. Kol and R. Ginosar, Adaptive Synchronization, Proc. of IEEE International Conference on Computer Design, 1998.

[8] R. Kol and R. Ginosar, Adaptive Synchronization, Proc. of Asynchronous Interfaces Workshop (AINT), TU Delft, The Netherlands, pp. 93-101, 2000.

[9] D.G. Messerchmitt, Synchronization in Digital System Design, IEEE Trans. Select. Areas Commun., Vol. 8, pp. 1404-1419, 1990.

[10] C.E. Cummings, Simulation and Synthesis Techniques for Asynchronous FIFO Design, SNUG 2002 .User Papers, 2002.

[11] U. Frank, T. Kapschitz and R. Ginosar, A Predictive Synchronizer for Periodic Clock Domains, Formal Methods in System Design (special issue on Formal Methods for Globally Asynchronous Locally Synchronous Design), 28(2), pp.171-186, 2004.

[12] S.R. Vangal, J. Howard *et al.*, An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS, IEEE Journal of Solid-State Circuits, 43(1), pp.29-41, 2008.

[13] F. Vitullo, N.E. L'insalata *et.al.*, Low-Complexity Link Microarchitecture for Mesochronous Communication in Networks-on-Chip, IEEE Trans. on Computers, Vol.57,no.9, pp. 1196-1201, 2008.

[14] S. Saponara, F. Vitullo *et.al.*, LIME: A Low-latency and Low-complexity On-chip Mesochronous Link with Integrated Flow Control, Proc. of the 11thEUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD), Parma, Italy, pp. 32-35, 2008.

[15] I.M. Panades and A. Greiner, Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures, Proc. of the 1st International Symposium on Networks-on-Chip (NOCS'07), pp.83-94, 2007.

[16] A. Chakraborty and M.R. Greenstreet, Efficient Self-Timed Interfaces for Crossing Clock Domains, Proc. of the 9th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'03),Vancouver, Canada, pp. 78-88, 2003.

[17] F. Mu and C. Svensson, Self-Tested Self-Synchronization Circuit for Mesochronous Clocking, IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing, Vol.48, no.2, pp. 129-141, 2001.

[18] B. Mesgarzadeh and C. Svensson, A New Mesochronous Clocking Scheme for Synchronization in SoC, Proc. of IEEE International Symposium on Circuits and Systems (ISCAS'04), Vancouver, Canada, pp. 605-609, 2004.

[19] D. Wiklund, Mesochronous Clocking and Communication in On-Chip Networks, Proc. of the Swedish System-on-Chip Conference (SSoCC'03), 2003.

[20] Y. Semiat and R. Ginosar, Timing Measurements of Synchronization Circuits, Proc. of the 9th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'03), Vancouver, Canada, pp. 68-77, 2003.

[21] W.J. Dally and S.G. Tell, The Even/Odd Synchronizer: A Fast, All-Digital, Periodic Synchronizer, Proc. of the IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC 2010), pp. 75-84, 2010.

[22] J.M. Chabloz and A. Hemani, Low-Latency and Low-Overhead Mesochronous and Plesiochronous Synchronizers, Proc. of the 2011 14th Euromicro Conference on Digital System Design, pp. 157-164, 2011.

[23] P. Caputa and C. Svensson, An On-Chip Delay- and Skew-Insensitive Multicycle Communication Scheme, Proc. of the IEEE International Solid-State Circuits Conference (ISSCC 2006), San Francisco, U.S, pp. 1765-1774, 2006.

[24] M. Ghoneima, Y. Ismail, M. Khellah and V. De, Variation-Tolerant and Low-Power Source-Synchronous Multi-Cycle On-Chip Interconnection Scheme, IEEE Trans. on Circuits and Systems I, Regular Paper, vol. 2007, pp. 1-12, 2007.

[25] T. Chelcea and S.M. Nowick, Robust Interfaces for Mixed-Timing Systems, IEEE Trans. on VLSI Systems, Vol.12, pp. 857-873, 2004.

[26] J. Jex and C. Dike, A fast resolving BiNMOS synchronizer for parallel processor interconnect, IEEE Journal of Solid-State Circuits, 30(2), pp. 133-139, 1995.

[27] M. Alshaikh, D. Kinniment and A. Yakovlev, A synchronizer design based on wagging, Microelectronics (ICM), International Conference on Microelectronics ,pp.415-418, 2010.

[28] S.R. Hasan, N. Belanger, Y. Savaria and M.O. Ahmad, All digital skew tolerant synchronous interfacing methods for high-performance point-to-point communications in deep sub-micron SoCs, Integration, the VLSI Journal 44.1, pp. 22-38, 2011.

[29] D. Ludovici, A. Strano *et.al.*, Comparing Tightly and Loosely Coupled Mesochronous Synchronizers in a NoC Switch Architecture, Proc. of the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOC2009), Salo, Finland, pp. 244-249, 2009.

[30] R.W. Horst, TNet: a reliable system area network, IEEE Micro, 1995.

[31] R. Ginosar, 2005, MTBF of a Multi-Synchronizer System on Chip, http://webee.technion.ac.il/~ran/papers/MTBFmultiSyncSoc.pdf, 2005

[32] E. Bolotin *et al.*, QNoC: QoS architecture and design process for network on chip, Journal of Systems Architecture 50.2, pp. 105-128, 2004.

[33] A. Strano, D. Ludovici and D. Bertozzi, A library of dual-clock fifos for cost-effective and flexible MPSoC design, Proc. Embedded Computer Systems (SAMOS), pp. 20-27, 2010.

[34] Xilinx, Core Generator, version 14.6.

[35] Altera, Mega Wizard, version 13.0.