INTEGRATION, the VLSI journal 42 (2009) 367–375

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Two-phase synchronization with sub-cycle latency

Rostislav (Reuven) Dobkin *, Ran Ginosar

VLSI Systems Research Center, Technion—Israel Institute of Technology, Haifa 32000, Israel

ARTICLE INFO

Article history:
Received 10 January 2008
Received in revised form
20 August 2008
Accepted 21 November 2008

Keywords:
Data synchronization
Asynchronous circuits
Clock domains
Two-phase protocol

ABSTRACT

Synchronizers typically incur long latency of multiple-clock cycles, resulting in low throughput. This paper presents two novel fast synchronizers, both based on two-phase protocols: a two-flip-flop synchronizer which reduces the data cycle from 6–12 down to 2–4 clock cycles, and a *LDL* synchronizer which strives for maximum throughput and 'sub-cycle latency,' namely data transfers that incur no extra penalty due to synchronization. These synchronizers are useful for data transfers over long interconnects. Simulations of best- and worst-case scenarios are presented which demonstrate the improved performance of the novel synchronizers. The results are compared to two-clock *FIFO* and to conventional two-flip-flop synchronizers.

1. Introduction

Large Systems on Chip (*SoCs*) typically integrate multiple-clock domains, stemming from interfacing different external frequencies, the integration of modules that were designed to operate on different frequencies, and clock gating and partitioning of large and fast clock trees. Moreover, in order to reduce power consumption, frequency and voltage may also be changed dynamically in *DVFS* systems [1–3], leading to changing clock relations during chip operation.

A *SoC* constructed of multiple-clock domains may be termed a globally asynchronous, locally synchronous (*GALS*) system [4,5]. This paper addresses the challenge of data synchronization and communication across clock domains in *GALS* systems. This challenge is further complicated by increasing global wire delays and increasing variability in those delays due to process variations and noise [6,7].

Asynchronous solutions for global communication across clock domains are preferred over synchronous ones since they eliminate the need for re-synchronization when crossing clock domains, do not require complex clock distributions and are more flexible under changing voltage and temperature conditions [8–12]. Thanks to these advantages, *ITRS* [13] predicts that by the year 2020, 40% of *SoC* global signaling will be performed asynchronously.

Dynamically changing clock frequencies and wire delay variations call for robust synchronizers that provide high data rate and low latency. The simple 'two-flip-flop' synchronizer typically incurs significant multi-cycle latency and limits throughput. An alternative solution is provided by two-clock *FIFO* synchronizers. However, they are intended only for cases when the two-clock domains are physically close to each other, because they are intolerant to delay variations over long wires. Further, they incur additional latency when the *FIFO* is empty. Other synchronizers employ stoppable clocks [14–21]. They must take into account additional latency due to clock tree delays [21,22,37].

Two synchronizers that employ two-phase protocols are presented in this paper: low-latency and sub-cycle latency synchronizers. The low-latency synchronizer employs two-flip-flop synchronization circuits, and is shown to minimize latency and enhance throughput relative to conventional two-flip-flop synchronizers.

The sub-cycle latency synchronizer proposed in this paper provides even higher throughput. It enables correct data sampling at the earliest possible edge of the sampling clock. The new synchronizer is based on locally delayed latching (*LDL*) [23], which is similar to the two-flip-flop synchronizer that does not require stopping of the clock. In contrast with the *LDL* synchronizer of [22,23], the synchronizers presented in this paper employ two-phase protocol over the communication channel and *enable* data transfers on each clock cycle. The proposed circuits employ standard interfaces, enabling seamless integration in modular *SoC* designs. The goal of the synchronizers presented in this paper is to enhance performance; power and area of the synchronizers are immaterial, because only a tiny fraction of total power and area are consumed by synchronizers in typical *SoCs*.

This paper considers the synchronization of mutually asynchronous clock domains, namely where the two clock frequencies are unrelated and could also change over time. Synchronizers which are optimized for mesochronous and multi-synchronous clock domains are treated elsewhere [24–26].

* Corresponding author. Tel.: +972 54 4248169.
E-mail address: rostikd@tx.technion.ac.il (R. Dobkin).

0167-9260/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2008.11.006

The paper is organized as follows. Section 2 compares low latency solutions, including novel two-phase two-flip-flop synchronizers, Section 3 presents sub-cycle latency *LDL* synchronizers, and simulations are described in Section 4.

2. Low-latency synchronizers

2.1. Two-flip-flop synchronizers

Standard asynchronous two-flip-flop synchronizers are widely employed [27–29]. The main assumption of such synchronizers is that the time reserved for metastability resolution provides a satisfactory mean time between failures (*MTBF*). The latency of the simple synchronizer is relatively high due to the time preserved for metastability resolution. This latency can be improved by sampling multiple times and employing speculative or non-speculative voting [30,31].

The chosen handshake protocol directly affects the synchronization data rate. The data rate can be improved by moving from a four-phase protocol to two-phase, especially in the case of long-range communication where wires incur additional high latency.

A simple two-phase synchronizer is shown in [29]. Fig. 1 shows a more aggressive design of the synchronizer. The synchronization circuit in the receiver clock domain (right-hand side) comprises *F*1, the *XOR* gate and the *Enable* input of *REGR*. The *XOR* gate and the toggle *F*2 convert the two-phase *REQ* into four-phase *RXE* and a single-cycle pulse *VO*. *F*4 provides for acknowledgement. The READY input facilitates back-pressure (the asynchronous input is not acknowledged if the receiver is busy).

The time reserved for metastability resolution is one clock cycle, minus the logic path delay from *F*1 to the *Enable* input of *REGR*. However, since the output of *F*1 branches to other targets, the resolution time is actually the clock cycle minus the maximum over the (bold, red) logic paths to *REGR*, *F*2, *F*3 and *F*4. All these paths should be constrained to as short a delay as possible. Otherwise, the synthesizer and/or physical design EDA tools may create longer logic and wire delays (as long as these delays are shorter than the clock cycle). Such extended delays may erode the time left for metastability resolution, and hence they should be eliminated by means of timing constraints. Similar constraints should be made for the synchronization circuit in the sender clock domain.

When fast clocks are used, a single-cycle time may be insufficient for reliable operation; the time for metastability resolution can then be extended by inserting additional flip-flops in front of *F*1 and/or *F*5.

The synchronizer operation is explained by the transmitter *FSM* in Fig. 2. Note that *TXS* (the *TX* state) is derived from the (bold, red) synchronization circuit and hence, its toggle time depends on metastability resolution, and can happen either one or two cycles after latching *F*5. The *TX FSM* accommodates this variability of toggling time by providing for either case. The output registers *REGD* and *REGV* are controlled by the *FSM* and by *TX Enable* (*TXE*), the resolving signal from the sampling flip-flop is marked in bold and red.

In mesochronous operation, the minimal data cycle time ($REQ+ \rightarrow REQ+$) is four clock cycles in the worst case, when the two clocks are in phase, and only three clock cycles when the clocks are out of phase. When the two clocks are mutually

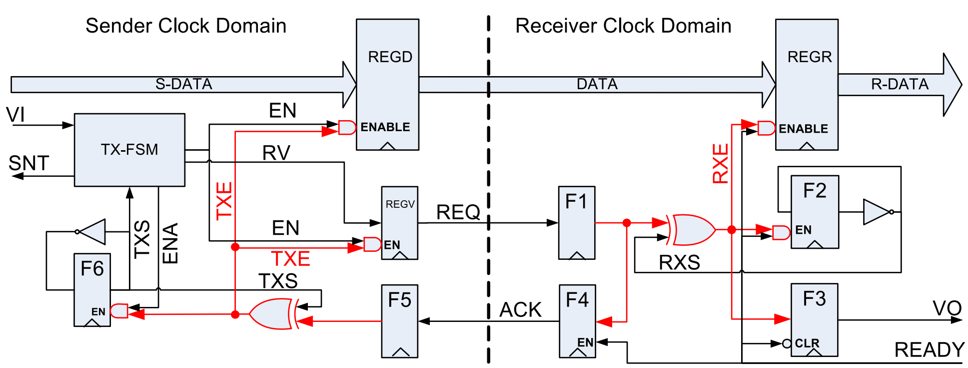

Fig. 1. Fast two-flip-flop two-phase synchronizer.

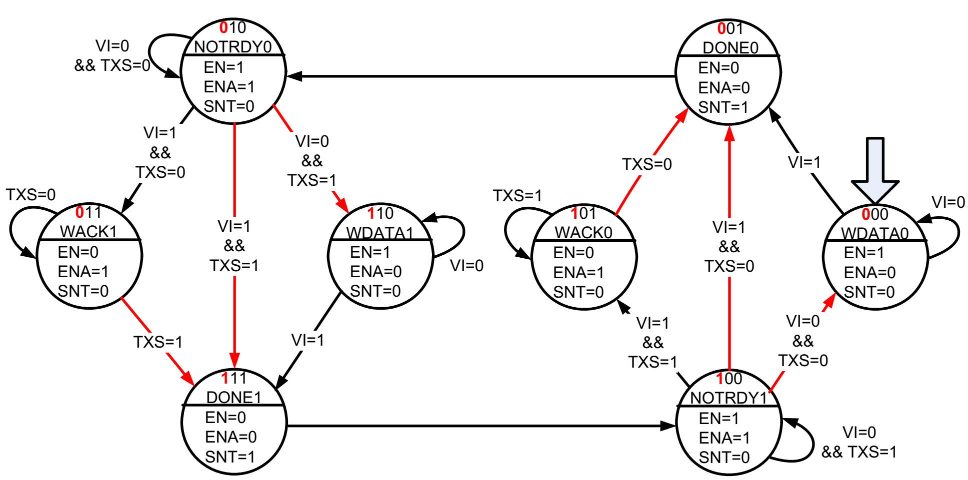

Fig. 2. TX FSM of the fast two-phase synchronizer.

Table 1
Data cycles of two-flip-flop synchronizers [32].

	Simple four-phase		Simple two-phase		Fast four-phase		Fast two-phase	
	Best (off-phase)	Worst (in-phase)	Best (off-phase)	Worst (in-phase)	Best (off-phase)	Worst (in-phase)	Best (off-phase)	Worst (in-phase)
Mesochronous clocks	10	12	4	6	4	6	3	4
Asynchronous clocks	$6 \cdot TX+6 \cdot RX$		$3 \cdot TX+3 \cdot RX$		$3 \cdot TX+3 \cdot RX$		$2 \cdot TX+2 \cdot RX$	

asynchronous, the data cycle depends largely on the slower clock and, if the clock ratio is larger than two, the data cycle is two clock cycles of the slower clock. Table 1 summarizes the data cycle figures for all cases and for simple and fast four- and two-phase synchronizers. The simple and the fast four-phase synchronizers are presented and analyzed in [32].

2.2. Two-clock FIFO

The two-clock *FIFO* synchronizer can transfer data on each clock cycle if the *FIFO* is neither full nor empty. The *FIFO*, however, is a more complex design that incurs higher data latency and does not support communication over long interconnect (at least one of the two communicating clock domains will have to be stretched over a long distance, making it impractical to maintain low skew at high frequencies). In [33], a mixed-timing *FIFO* was proposed for communication between arbitrary combinations of synchronous and asynchronous domains. Mixed timing relay stations were also introduced for more efficient treatment of long interconnects. Source–synchronous communication, based on a self-timed single-stage *FIFO* with a single stage for mesochronous clock domains was presented in [34] and expanded to multi-synchronous, plesiochronous and asynchronous cases in [35]. The extensions are more complex relative to the mesochronous case, requiring additional special treatment at the transmitter and receiver sides.

2.3. Stoppable clocks

Data synchronization can be also performed by controlling the capturing clock. Stoppable local clocks technique was proposed for GALS systems in [15–20,36–38]. The technique incorporates a local ring-oscillator clock generator in each synchronous ‘island’ with a set of *MUTEXes* [39] that stop the clock temporarily when new input data arrive. Handshake clocks [14] can be employed, stopping the capturing clock based on inputs from other domains. A stoppable clock technique suitable for linear pipelines was presented in [21]. In order to achieve performance enhancement, stoppable clock techniques are sometimes accompanied by *FIFOs* [15,36].

2.4. Categorizing synchronizers

Based on the works listed above, we categorize the synchronization approaches into a number of simple cases (Fig. 3). When the transmitter and receiver belong to the same clock domain and are placed close to each other, no synchronization is required (a). A *FIFO* can be inserted for additional buffering (b). Fast synchronizers should be employed when transferring data between different clock domains, to enable high throughput and low latency and to reach as much as possible the performance of intra-clock domain transfers as in (a).

A two-clock *FIFO* (c) may achieve high data rates, but it incurs higher latency. Stoppable-clock synchronizers (d) communicate with local clock generators to minimize the latency, and *LDL* synchronizers (also described by d) control the synchronous interfaces by introducing dynamic local clock delays. For long-range interconnect, wire delays degrade significantly the data cycle regardless of the synchronization circuits speed. In order to improve the throughout, pipelining can be employed along the link (e). The pipeline can be either synchronous (with *TX* or *RX* clocks) or asynchronous.

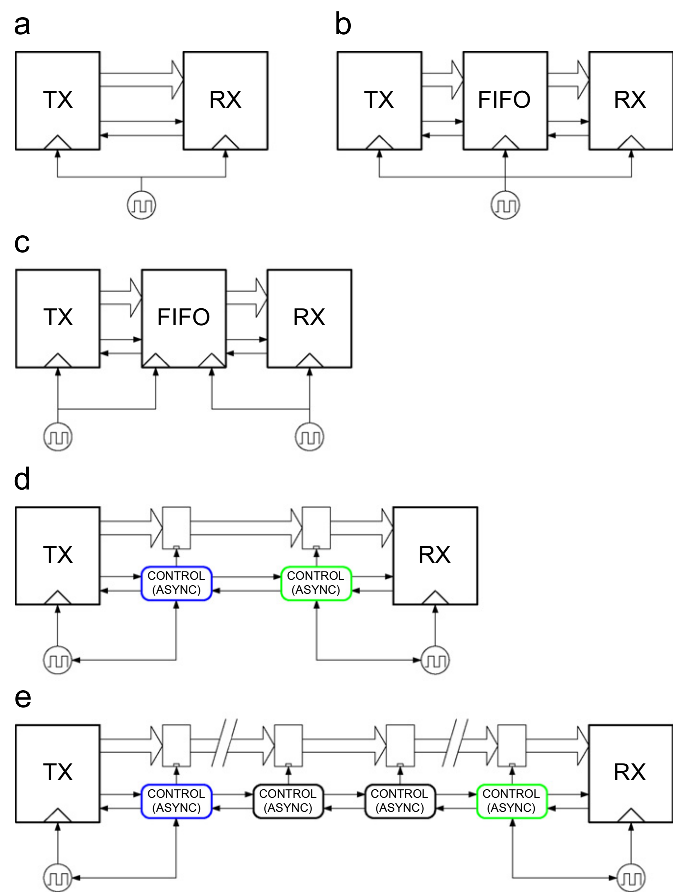

Fig. 3. Fast synchronization: (a) single clock domain transfer, (b) buffered single clock domain transfer, (c) two-clock *FIFO* synchronizer, (d) stoppable clock and *LDL* synchronizers and (e) synchronization over long interconnect.

To facilitate modularity and ease of integration, the transmitter and receiver should not be aware of the synchronizer and should provide standard interfaces, not only in (a)–(c) but also in (d)–(e) of Fig. 3.

3. Sub-cycle latency synchronization

In our previous works [22,23], the *LDL* synchronizers employed four-phase protocols and the ports did not support data transfer on each clock cycle. Four-phase protocols incur high latency,

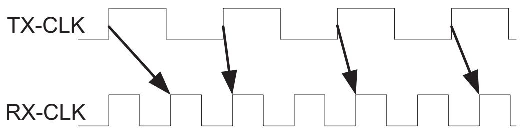

Fig. 4. Sub-cycle latency synchronization.

especially over long links. In this work we present new *LDL* architecture which overcomes these obstacles. The two-phase version of the *LDL* synchronizer can achieve sub-cycle latency. The section starts with definitions (including the meaning of 'sub-cycle' latency), presents *LDL* concepts, and describes *LDL* input and output.

3.1. Definitions

The *forward latency* of a synchronizer is defined as the time from writing a data word into the output register of the sender (*TX*) to writing the same data word into the first register of the receiver (*RX*), namely it is the time for moving the data from the *TX* to *RX* clock domain. The *data cycle* is the time between two successive writings of the first register of the receiver. *Throughput* (in data words) is the inverse of the data cycle. The data cycle and throughput of intra-clock domain transfer are the clock cycle and the clock frequency, respectively, and we wish to attain similar cycle and throughput in fast synchronizers. Typically, such *sub-cycle latency synchronizers* incur latency less than a single clock cycle (of the slower clock), managing to latch the data safely into *RX* on the earliest clock edge, imitating the latency of intra-clock domain transfers. Fig. 4 exemplifies the timing of such a synchronizer: the *TX* data is sampled on the first *RX* clock following the transfer.

Interconnect delay affects both latency and throughput of the synchronizer. The latency is extended by the delay, and the data cycle is extended by four and two times the interconnect delay in four- and two-phase synchronizers, respectively. Pipelining the communication link reduces this data cycle penalty at the expense of additional latency.

3.2. Locally delayed latching

Locally delayed latching [23] does not require stopping the clock of locally synchronous islands. *LDL* is unaffected by any dynamic scaling of the clock cycle [1–3]. An asynchronous input port (Fig. 5) controls both the input latch and *Y*1, the clock input to the first sampling register. The local clock *Y* is uninterrupted. The port issues a valid indication for each new data word, prevents write-after-read hazards and can be stopped when the locally synchronous island is not ready to receive data.

Instead of stopping the clock *Y*, *Y*1+ is delayed when a conflict is imminent. *Y*1− is unaffected: only the high-phase is shortened (see Fig. 6). The worst case occurs when the incoming *REQ* conflicts with clock *Y* and *REQ* wins the arbitration, possibly after the metastability resolution time *M*/*S*. In this case, the high-phase of *Y*1 (*HP*) is maximally shortened. For other cases, the high-phase of *Y*1 is either shortened by a smaller extent or not shortened at all [22]. The shorter cycle leaves less time for computing in the combinational logic immediately following the first register (*CL* in Fig. 5).

In effect, the time for metastability resolution is borrowed from the next clock cycle. Consequently, *LDL* poses several timing restrictions [22]. Once metastability is resolved, the controller latches the incoming data by pulsing *L* for D_{CTRL} time. All this must complete at least HP time before *Y*−, so that the clock *Y*1 can be

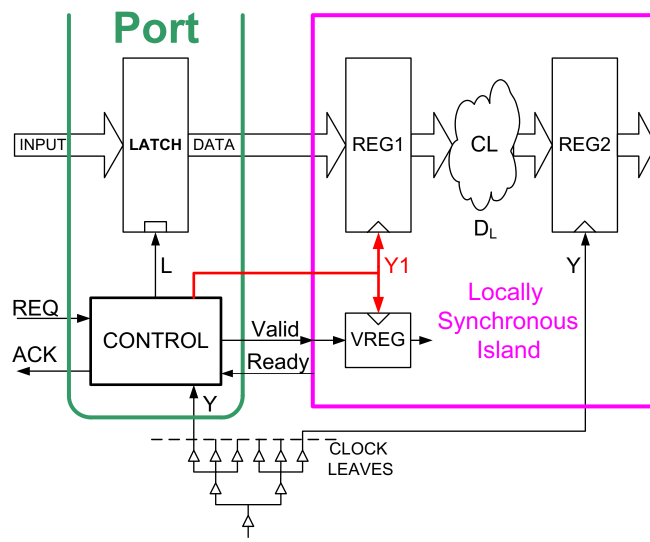

Fig. 5. *LDL* input port connected to a synchronous island.

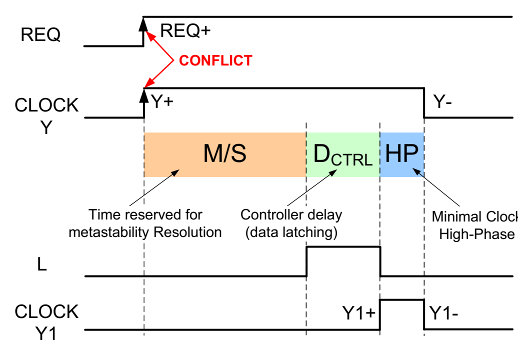

Fig. 6. *LDL* time budgets for worst-case operation.

high for at least *HP* time (the minimal high-phase width required by the technology, usually 2–3 gate delays). Note that HP also contains the delay of the gated clock tree *Y*1 and may be affected by word-width. Thus, the minimal high phase of the *Y* clock is bounded by $M/S+D_{CTRL}+HP$. In case of a symmetric clock, this defines the minimal half-clock cycle of *Y*. Note that *M*/*S* and *HP* are system requirements, while D_{CTRL} depends on the implementation of the controller. In addition, as explained above, in case of conflict the time D_L available for computation by the combinational logic (*CL* between *REG*1 and *REG*2) is shortened down to half a cycle [22]. Thus, that logic stage should be constrained to a delay of less than half a cycle.

The metastability resolution time requirement is derived from global *SoC MTBF* requirements. Assuming *SoC MTBF* requirement of 100 years and 100 synchronizers in the *SoC*, the desired *MTBF* of a single synchronizer should be 10,000 years [40]. This requirement can be achieved if the *M*/*S* period in Fig. 6 is at least 43τ [22], where τ is the metastability resolution time constant and is assumed to be about one *FO*4 gate delay [41]. Typical *SoCs* employ clock cycle times *T* in the range 100–400 *FO*4 gate delays [13]. For $T = 100$ *FO*4, metastability resolution period of 43τ takes almost one half of a symmetric clock cycle. For slower *SoCs*, e.g. where the fastest clock cycle is 170τ, a quarter clock cycle suffices to achieve this MTBF. *LDL* can support faster clocks than $T = 100\tau$ by extending the total time budget (changing the duty cycle by enlarging the relative portion of the high phase) using low-complexity minimal phase generation circuit [22]. For more aggressive designs (such as high-speed processors or high-speed *ASIC* modules) where $T<50\tau$, a modified approach based on multi-cycle resolution time or on multi-synchronous clocking is required [25,26].

The said three time-intervals (Fig. 6) and interconnect delay between *TX* and *RX* are the main parameters influencing *LDL* synchronization performance in terms of latency and throughput. In addition, standard interfaces are required at the transmitter and the receiver in order to support seamless integration into standard *HDL* design. In the following, a *FIFO*-like interface is employed.

In the *LDL* synchronization the data is sampled at the closest receiver clock edge when there is no conflict between the data and clock. Hence, this scheme supports sub-cycle latency requirements. When a conflict occurs, a single clock cycle penalty is paid only in statistically half of the conflict cases.

3.3. Two-phase LDL input port

Fig. 7 shows the *LDL* input port. An asynchronous controller provides for the *REQ/ACK* handshake and for control of the *MUTEX*. When the controller has data, it raises *ASK*, which eventually leads to *LATCHED*+ that captures the new data into *REG*. At that time, *Y*1 is kept low.

When the receiver is not ready to accept a new data word, *READY* signal is de-asserted, blocking new data latching into *REG*1 and blocking de-assertion of *VALID*. Note that even when the *READY* is low, the *LDL-IP* can latch a new data word into *REG*, preparing it for the next valid synchronization cycle. However, once a new data word is latched inside *REG*, the next handshake is blocked by *VALID* = high.

The two-phase protocol minimizes the penalty caused by the interconnect delay. The controller signal transition graph (*STG*) [44], circuit implementation and example waveforms are shown in Fig. 8. Note the *D-FFs* that are wired to operate as toggle (*T*)

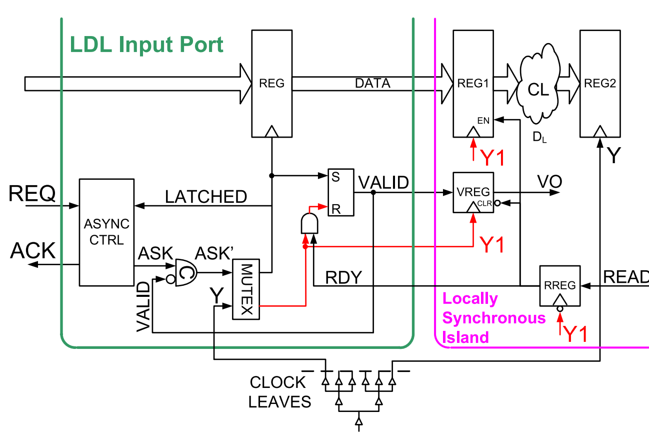

Fig. 7. *LDL* input port.

elements. This simple circuit employs only standard library cells. It converts the two-phase *REQ/ACK* protocol into the four-phase *ASK/LATCHED* protocol. The circuit assumes that the delay for the path *ASK*+→*LATCHED*+ should be longer than *ASK*+→*T*→*S*–, to avoid set–reset conflict. This condition is easily met thanks to the port structure.

3.4. Two-phase LDL output port

The output port is shown in Fig. 9, in addition to special interface circuitry within the synchronous island the output port interfaces the synchronous island by two signals, similar to a standard *FIFO* handshake: *VALID* and *FULL*. Upon *FULL*, additional data are stalled. In addition, the output port generates the possibly delayed clock *Y*1.

The *STG* of the asynchronous controller of the output port, its circuit and example waveforms are shown in Fig. 10. The circuit converts the four-phase *RI/ASK* protocol to the two-phase *RO/AO*. The controller delay D_{CTRL} consists of the delay of the *C*-element (*R*+→*RI*– in Fig. 9) and the internal controller delay *RI*–→*ASK*– (a single gate delay). The critical delay of the output port is very similar to that of the input port. The circuit employs the timing assumption similar to the one of the input port.

The output port must synchronize *FULL*, as specified in Fig. 11: the assertion of *VALID* on *Y*1+ sets *FULL*, blocking the transmission of the next word. *FULL* is de-asserted following the toggle of *AKIN* and only during the low-phase of the clock *Y* (also *Y*1), thus preventing contention at the sampling register *REGF*.

Operation is demonstrated in Fig. 13. For each new data, *VALID* is asserted, leading to raising *FULL* and self-resetting *VALID* (*AR* = asynchronous reset in Fig. 9). If the targeted input port acknowledges (toggling *AKIN*) within a single *TX* clock cycle (de-asserting *FULL*), new data can be sent on the next clock cycle (case ♯1 in Fig. 13). Thus, data can be transferred on each clock cycle of *TX*. When *FULL* is high during the rising edge of *TX* clock *Y*1, the data is not changed (output flip-flops are disabled, case ♯2 in Fig. 13). In the intermediate situation, when the incoming acknowledge contends with clock *Y*, there are two possible cases. In the first case, clock *Y* wins over the acknowledge signal *ASK* and therefore *FULL* is de-asserted only on the next falling edge of *Y* (one clock penalty in sending data, case ♯3 in Fig. 13). In the second case, the acknowledge signal *ASK* wins over clock *Y*. Then, *FULL* is de-asserted and later on clock *Y*1 is unblocked, resulting in shortened *Y*1 cycle (case ♯4 in Fig. 13). Note that in both ♯3 and ♯4 cases we have to retain the next data over the normal clock edge, which is obtained by clocking *REGD* and *REGV* registers by *Y*1. In the following we discuss implementation issues of the output port.

The *LDL* output port of Fig. 9 requires making two timing assumptions as follows:

*TA*1: Whereas both *E*+ (the *Enable* of *REGD*, *REGV*) and *Y*1+ emanate from *FULL*–, the former must precede the latter (Eq. (1)).

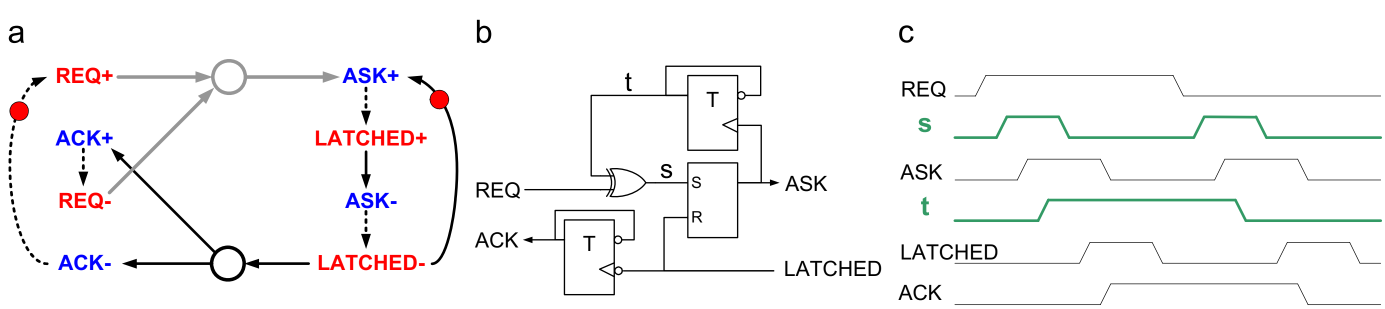

Fig. 8. Two-phase input-port asynchronous controller.

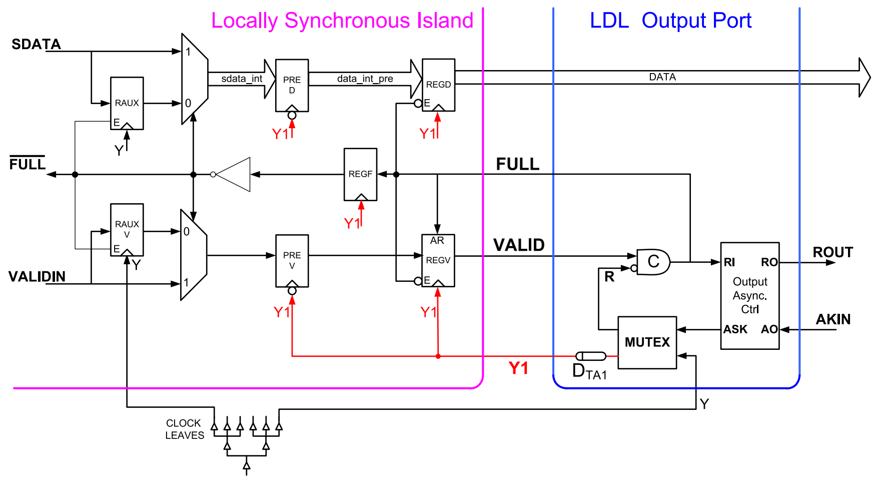

Fig. 9. *LDL* output port.

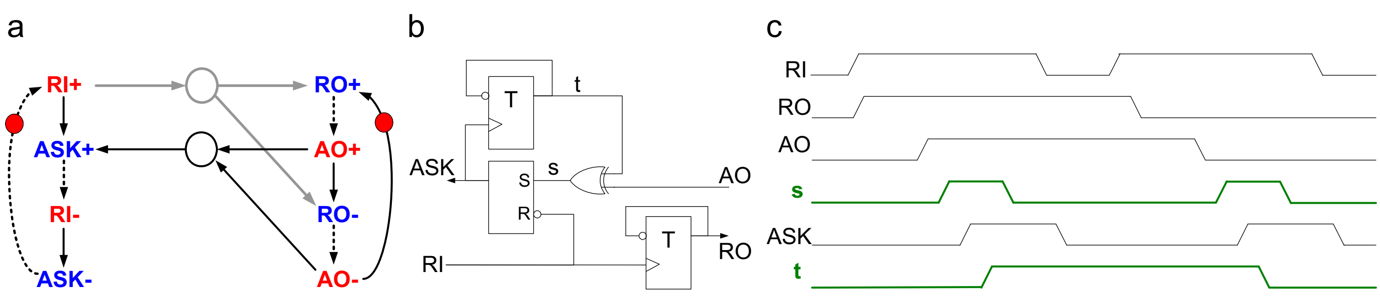

Fig. 10. Two-phase output-port asynchronous controller.

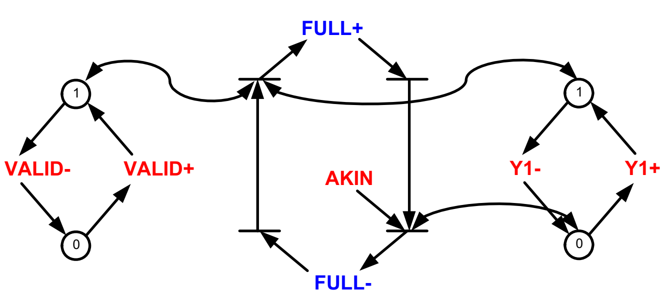

Fig. 11. *FULL* signal generation.

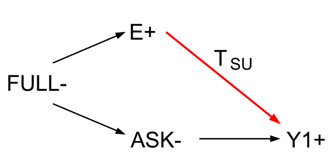

Fig. 12. Timing assumption ♯1.

This requirement is easily met in the circuit ($D_{TA1}=0$ in Fig. 9) (Fig. 12).

$$TA1:\ Delay(FULL- \rightarrow E+) < Delay(FULL- \rightarrow ASK- \rightarrow Y1+) - T_{SU} \quad (1)$$

TA2: Hold time T_H should be satisfied for registers REGD and REGV. Clock *Y1* is phase shifted relative to *Y* by the MUTEX metastability resolution time in cases of contention between *Y* and *ASK* (case ♯3 in Fig. 13). Note that the skew incurred by non-conflict *MUTEX* delay is eliminated by balancing *Y* and *Y1* clock trees. In the case of contention, the skew between *Y* and *Y1* can be as large as T_{MUTEX}+D_{CTRL}. Thus

$$TA2:\ T_H > T_{MUTEX} + D_{CTRL} \quad (2)$$

The value of T_{MUTEX}+D_{CTRL} is very high and therefore hard to meet by simple delay line. As shown below, *TA2* is achieved by adding a negative-triggered *FFs* (*PRE-D* and *PRE-V*).

The sub-cycle latency *LDL* synchronizer supports throughput of one data item per cycle (*DPC*). To enable that, the synchronous interface must be able to supply data words at the highest rate, while satisfying the timing assumptions above. *SDATA* (Fig. 9) may carry new data only if *FULL* was low at the previous clock cycle. When *FULL* is set, *REGD* contains a data item, a second data item is provided on *SDATA* but cannot be stored into *REGD*, and the synchronous pipeline behind may be ready to overwrite *SDATA* on the next rising edge of *Y*. To avoid loss of the word currently on *SDATA*, auxiliary registers [42] *RAUX* and *RAUXV* are added (Fig. 9). The hold requirement (2) is satisfied by the negative edge triggered *PRE_D* and *PRE_V* registers, which stabilize the inputs of *REGD*, *REGV* during the next *Y1*+, even when *Y1* is delayed relative to *Y* and the data on *SDATA* has changed. Note that *SDATA* and *VALIDIN* must be valid within half a cycle, which is easily met.

Consider the timing of *PRE-D* and *REG-D*. *PRE-D* is latched one half-cycle after the *FFs* that precede it (*Y1*− is not delayed relative to *Y*−). Thus, it is possible to insert logic that requires half a cycle before *PRE-D*. Similarly, *REGD* is latched no sooner than half a cycle after *PRE-D*, allowing the insertion of logic between them. Thus, this interface circuit allows useful work and does not incur any idle latency. In cases of conflict, in approximately half the cases (when *Y* wins at the *MUTEX*) single-cycle latency is inserted. However, in the other half of the cases (*ASK* wins at the *MUTEX*)

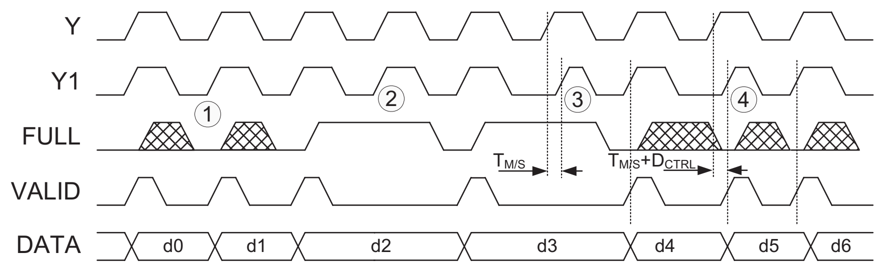

Fig. 13. *LDL* output port operation.

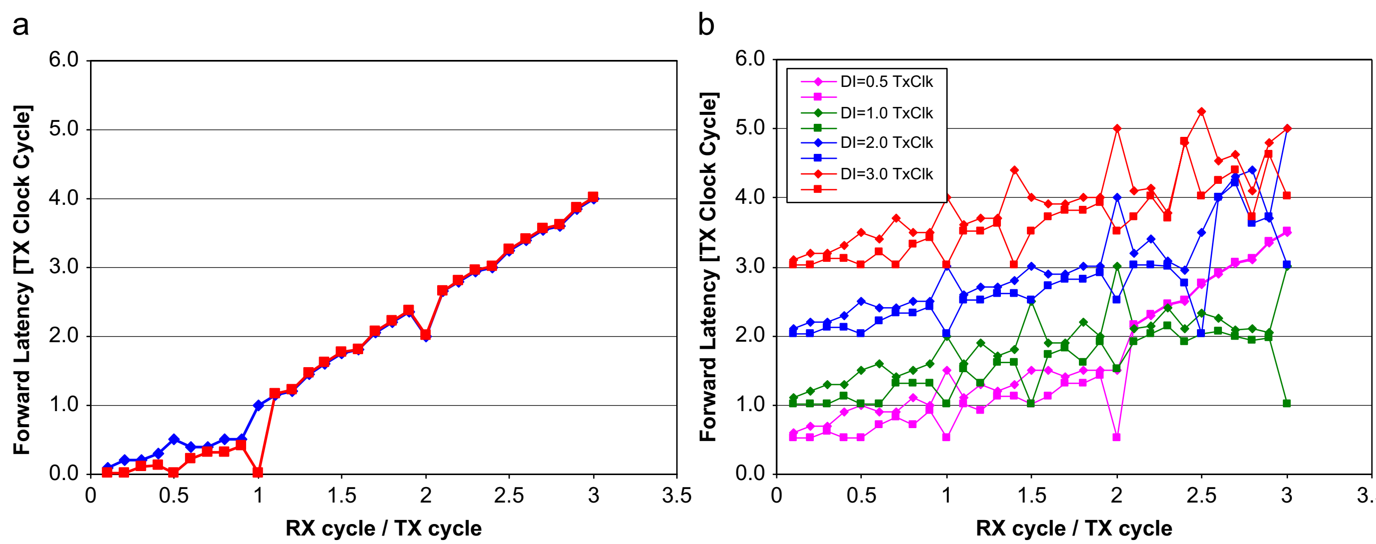

Fig. 14. *LDL* synchronization latency bounds.

there is no added latency and no loss of throughput. Another solution for the hold requirement, requiring no negative-triggered *FFs*, is presented in [32].

The proposed *LDL* synchronizer contains standard logic cells (both *MUTEX* and *C*-element can be implemented by standard cells, although it is better to add these cells to the technology library), and gated and inverted clocks. Furthermore, all timing constraints required for the design of the *LDL* interfaces can be specified as normal constraints for standard synthesis, *STA* and place-and-route *EDA* tools. The asynchronous input and output ports are also synthesizable by standard *EDA* tools, and no special asynchronous design tools are required.

4. Performance simulations

In this section we compare the performance of the sub-cycle latency *LDL* synchronizer with the fast two-phase two-flip-flop synchronizer of Section 2.1 and with a standard two-clock *FIFO* synchronizer [43]. A *FIFO* depth of 10 and bursts of 1000 words were employed. The analysis is not limited to any specific fabrication process, since it is based on cycle time ratios, and on scaleable measures such as the number of *FO*4 gate delays per clock cycle in *SOCs*. Thus, the results depend only on architecture. Note also that only performance measures (latency and data rate) are discussed; power and area are ignored, since only a tiny fraction of total power and area are consumed by synchronizers in typical *SoCs*.

4.1. Forward synchronization latency

Fig. 14(a) shows upper and lower bounds of the *LDL* synchronizer latency for back-to-back connection (no intercon-

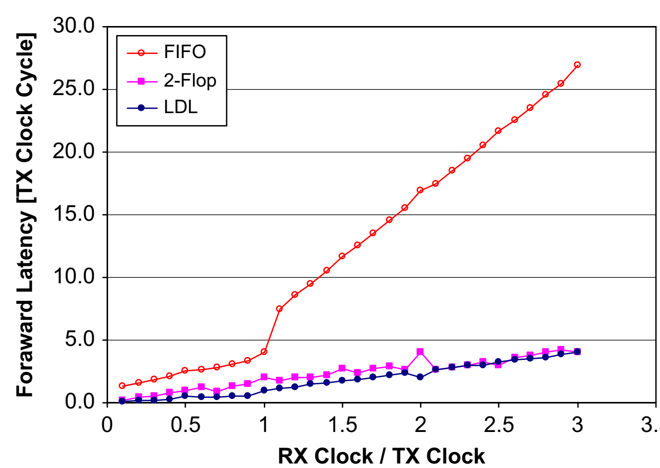

Fig. 15. Latency of two-flop, two-clock *FIFO* and *LDL* synchronizers.

nect delay). The latency is lower than half a cycle when *RX* clock is faster than *TX*, clearly indicating sub-cycle latency. When *RX* clock is slower than *TX* (higher than 1 on the horizontal axis), the latency grows with the *RX* clock cycle. In (b), the upper and lower bounds are shown for different interconnect delays D_I. Note that the lower bound is always limited by the interconnect delay. Note that mesochronous (same frequency) and periodic (integral frequency ratio) clocks result in increased latency difference between best and worst cases, as well as other special values of the ratio of cycle times [32].

Fig. 15 compares two-flip-flop, *FIFO* and *LDL* synchronizers. The latency of the *FIFO* is the longest, and the *LDL* synchronizer incurs the least latency. Note that the two-flip-flop synchronizer experiences worst case performance for integral clock relationship (e.g. $RX/TX = 2$), whereas the *LDL* synchronizer does not [32].

The *LDL* synchronizer outperforms two-flip-flop synchronizers even when interconnect delays are considered (Fig. 16). Note that the *FIFO* is not included in this comparison since a standard *FIFO* is not suitable for operation in the presence of long interconnect delays.

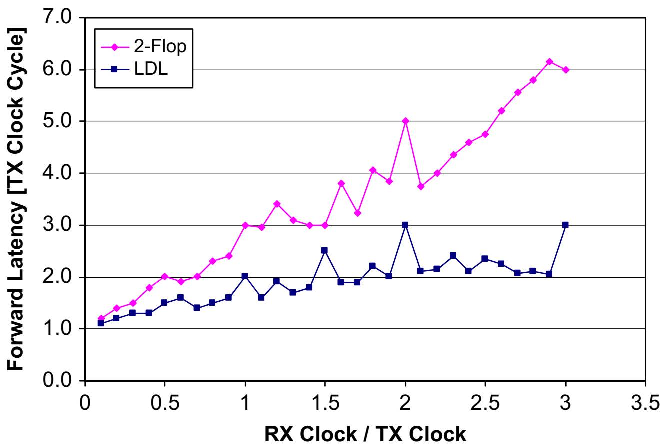

Fig. 16. Latency of *LDL* and two-flop synchronizers for $1.0 \cdot$ *TX-CLK* interconnect delay.

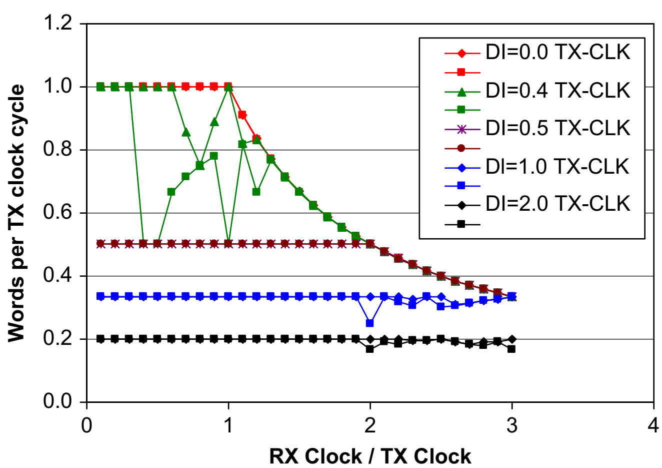

Fig. 17. *LDL* synchronization throughput bounds.

4.2. Data rate

Lower and upper bounds of data rate of the *LDL* synchronizer for different interconnect delays are shown in Fig. 17. Note that the theoretical lower bound of the data cycle is twice the interconnect delay, namely the flight time of *REQ* and *ACK*. To the right of '1', the upper bound is hyperbolic as the data rate is bounded by the inverse of *RX* cycle. The level lines demonstrate the effect of D_I. Only in certain cases the upper and lower bounds differ, similarly to Fig. 14. These differences originate from certain periodic relations of the clocks, as explained in [32].

LDL and *FIFO* throughputs are similar as evident from the overlapping charts in Fig. 18 and are about twice faster than the fast two-flip-flop two-phase synchronizer (as expected, see Table 1). As the interconnect delay grows, the synchronizer overhead becomes relatively smaller, and the synchronizers converge to relatively similar performance, but *LDL* always outperforms the two-flip-flop synchronizer.

5. Conclusions

Two novel synchronizers that employ two-phase protocols have been presented: a low-latency two-flip-flop and a sub-cycle latency *LDL* synchronizers. They facilitate clock domain crossings both when the two domains are physically adjacent and when they are separated by long interconnect.

The low-latency two-phase two-flip-flop synchronizer is shown to introduce only minimal latency, and to enable short data cycles of 2–4 clock cycles, compared to 6–12 clock cycles of a simple two-flip-flop synchronizer.

The two-phase *LDL* synchronizer is significantly faster than its four-phase predecessor [22,23]. It is termed 'sub-cycle latency' because it does not add any latency penalty (relative to synchronous data transfers) when crossing clock domains, and it enables sending data every clock cycle (similar to synchronous data transfers). The *LDL* sub-cycle latency synchronizer consists of asynchronous input and output ports, and certain modifications of the synchronous islands of a *GALS* system. The *LDL* synchronizer outperforms standard synchronizers (*FIFO* and two-flip-flop) in

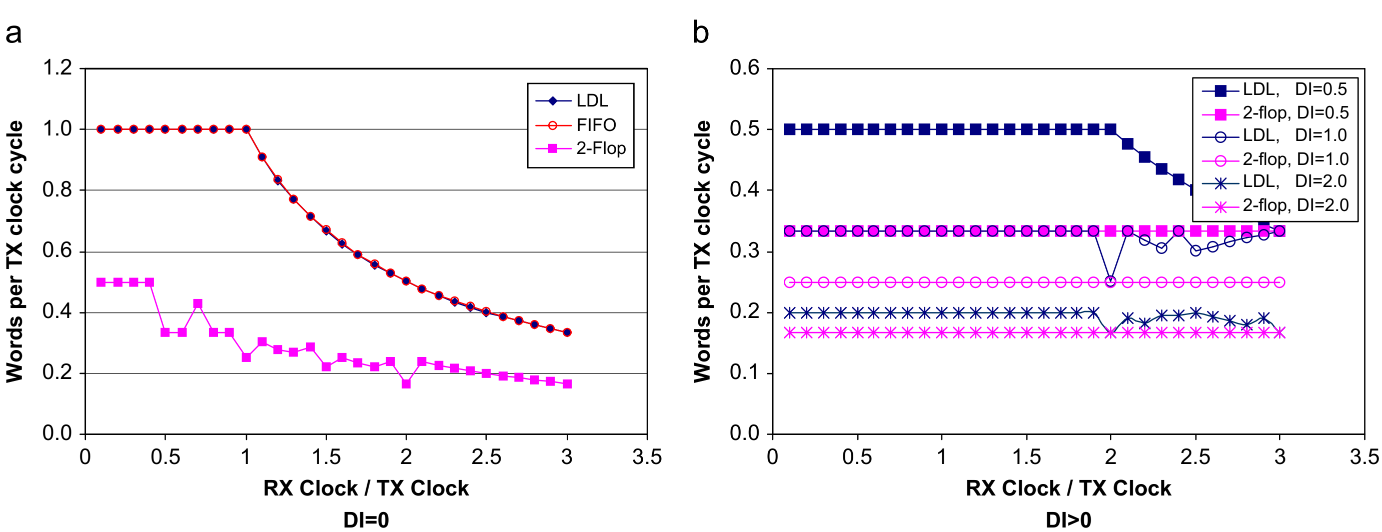

Fig. 18. Throughput comparison for *LDL*, two-flop and *FIFO* synchronizers.

terms of latency and throughput. The presented circuits have standard interfaces and require standard logic cells, thus enabling straightforward integration into standard designs.

References

[1] G. Semeraro, D.H. Albonesi, S.G. Dropsho, G. Magklis, S. Dwarkadas, M.L. Scott, Dynamic frequency and voltage control for a multiple clock domain microarchitecture, in: IEEE/ACM International Symposium on Microarchitecture, 2002, pp. 356–367.
[2] L.S. Nielsen, C. Niessen, J. Sparsø, C.H. van Berkel, Low-power operation using self-timed and adaptive scaling of the supply voltage, TVLSI 2 (4) (1994) 391–397.
[3] W.R. Daasch, C.H. Lim, G. Cai, Design of VLSI CMOS circuits under thermal constraint, TVLSI 49 (8) (2002) 589–593.
[4] D.M. Chapiro, Globally-asynchronous locally-synchronous systems, Ph.D. Dissertation, Stanford University, 1984.
[5] D. Bormann, P. Cheung, Asynchronous wrapper for heterogeneous systems, Proc. ICCD (1997) 307–314.
[6] L. Scheffer, An overview of on-chip interconnect variation, SLIP (2006) 27–28.
[7] R.O. Topaloglu, A.B. Kahng, Generation of design guarantees for interconnect matching, SLIP (2006) 29–34.
[8] J. Bainbridge, S. Furber, Chain: a delay-insensitive chip area interconnect, IEEE Micro 22 (5) (2002) 16–23.
[9] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, M. Renaudin, An asynchronous NOC architecture providing low latency service and multi-level design framework, ASYNC (2005) 54–63.
[10] T. Felicijan, S.B. Furber, An asynchronous on-chip network router with Quality-of-Service (QoS) support, Int. SOC Conf. (2004) 274–277.
[11] T. Bjerregaard, J. Sparso, A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip, ASYNC (2005) 34–43.
[12] R. Dobkin, V. Vishnyakov, E. Friedman, R. Ginosar, An asynchronous router for multiple service levels networks on chip, ASYNC (2005) 44–53.
[13] International Technology Roadmap for Semiconductors (ITRS), 2003–2005, http://www.itrs.net⟨www.itrs.net⟩.
[14] J. Kessels, A. Peeters, P. Wielage, S.J. Kim, Clock synchronization through handshake signaling, ASYNC (2002) 59–68.
[15] S. Moore, G. Taylor, R. Mullins, P. Robinson, Point to point GALS interconnect, ASYNC (2002) 69–75.
[16] S. Oetiker, F.K. Gürkaynak, T. Villiger, H. Kaeslin, N. Felber, W. Fichtner, Design flow for a 3-million transistor GALS test chip, ACiD Workshop (2003).
[17] T. Villiger, H. Kaeslin, F.K. Gürkaynak, S. Oetiker, Wolfgang Fichtner, Self-timed ring for globally-asynchronous locally-synchronous systems, ASYNC (2003) 141–150.
[18] J. Muttersbach, T. Villiger, W. Fichtner, Practical design of globally-asynchronous locally-synchronous systems, ASYNC (2000) 52–61.
[19] K.Y. Yun, R.P. Donohue, Pausible clocking: a first step toward heterogeneous systems, ICCD (1996) 118–123.
[20] K.Y. Yun, R.P. Donohue, Pausible clocking-based heterogeneous systems, TVLSI 7 (4) (1999) 482–488.
[21] A.E. Sjogren, C.J. Myers, Interfacing synchronous and asynchronous modules within a high-speed pipeline, TVLSI 8 (5) (2000) 573–583.
[22] R. Dobkin, R. Ginosar, C.P. Sotiriou, High rate data synchronization in GALS SoCs, TVLSI 14 (10) (2006) 1063–1074.
[23] R. Dobkin, R. Ginosar, C. Sotiriou, Data synchronization issues in GALS SoCs, ASYNC (2004) 170–179.
[24] Y. Semiat, R. Ginosar, Timing measurements of synchronization circuits, ASYNC (2003) 68–77.
[25] R. Kol, R. Ginosar, Adaptive synchronization, ICCD (1998) 188–189.
[26] D.J. Kinniment, Synchronization and Arbitration in Digital Systems, Wiley, New York, 2008.
[27] R. Ginosar, Fourteen ways to fool your synchronizer, ASYNC (2003) 89–96.
[28] W.J. Dally, J.W. Poulton, Digital Systems Engineering, Cambridge University Press, Cambridge, UK, 1998.
[29] N.H.E. Weste, D. Harris, CMOS VLSI Design, third ed., Addison-Wesley, Reading, MA, 2005.
[30] D.J. Kinniment, A. Yakovlev, Low latency synchronization through speculation, PATMOS (2004) 278–288.
[31] S.J. Kim, J.G. Lee, K. Kim, A parallel flop synchronizer for bridging asynchronous clock domains, AP-ASIC (2004) 184–187.
[32] R. Dobkin, R. Ginosar, Zero phase latency synchronizers using four and two phase protocols, CCIT TR642, EE Publication No. 1599, EE Department, Technion, 2007, http://www.ee.technion.ac.il/~ran/papers/zerolatency.pdf ⟨www.ee.technion.ac.il/~ran/papers/zerolatency.pdf⟩.
[33] T. Chelcea, S.M. Nowick, Robust interfaces for mixed-timing systems, TVLSI 12 (8) (2004) 857–873.
[34] A. Chakraborty, M.R. Greenstreet, Efficient self-timed interfaces for crossing clock domains, ASYNC (2003) 78–88.
[35] A. Chakraborty, M.R. Greenstreet, A minimal source–synchronous interface, ASIC/SOC (2002) 443–447.
[36] S. Chakraborty, J. Mekie, D.K. Sharma, Reasoning about synchronization techniques in GALS systems: a unified approach, FMGALS (2003).
[37] J. Mekie, S. Chakraborty, D.K. Sharma, Evaluation of pausible clocking for interfacing high speed IP cores in GALS framework, VLSI Des. (2004) 559–564.
[38] R. Mullins, S. Moore, Demystifying data-driven and pausible clocking schemes, ASYNC (2007) 175–185.
[39] C.L. Seitz, System timing, in: C.A. Mead, L.A. Conway (Eds.), Introduction to VLSI Systems, Addison-Wesley, Reading, MA, 1980 (Chapter 7).
[40] R. Ginosar, MTBF of multi-synchronizer SoC, ⟨http://www.ee.technion.ac.il/~ran/papers/MTBFmultiSyncSoc.pdf⟩.
[41] C. Dike, E. Burton, Miller and noise effects in a synchronizing flip-flop, JSSC 34 (6) (1999) 849–855.
[42] L. Carloni, A. Sangiovanni-Vincentelli, Coping with latency in SoC design, IEEE Micro (special issue on SoC) 22 (5) (2002) 24–35.
[43] Synopsys Design Ware FIFO, http://www.synopsys.com/products/designware/docs/doc/dwf/datasheets/dw_fifo_s2_sf.pdf⟨www.synopsys.com/products/designware/docs/doc/dwf/datasheets/dw_fifo_s2_sf.pdf⟩.
[44] T.A. Chu, C.K.C. Leung, T.S. Wanuga, A design methodology for concurrent VLSI systems, in: Proceedings of the ICCD, 1985, pp. 407–410.

Rostislav (Reuven) Dobkin received his B.Sc. and M.Sc. degrees in Electrical Engineering from the Technion—Israel Institute of Technology in 1999 and 2003, respectively. He is currently pursuing his Ph.D. in the same institute. Through the years 1997–2000 he worked within RAFAEL ASIC Experts design group and between 2001 and 2002 he lead a VLSI design group in IC4IC LTD., developing family of chips for communications. In parallel, Reuven served as a teaching assistant at the Technion EE Department since 1999. His research interests are VLSI architectures, parallel architectures, asynchronous logic, high-speed interconnect, synchronization, GALS systems, NoC.

Ran Ginosar received his B.Sc. from the Technion in 1978 and his Ph.D. from the Princeton University in 1982, both in Electrical and Computer Engineering. He worked at the AT&T Bell Laboratories in 1982–1983, and joined the Technion in 1983. He was a visiting Associate Professor with the University of Utah in 1989–1990, and a visiting faculty with Intel Research Labs in 1997–1999. He co-founded five companies in the areas of electronic imaging, medical devices, and wireless communications. He is an Associate Professor at the Department of Electrical Engineering and Computer Science and he serves as Head of the VLSI Systems Research Center at the Technion. His research interests include VLSI architecture, asynchronous logic and synchronization, networks on chip, electronic imaging, and neuro-processors.