CCIT Report #642 October 2007

Zero latency synchronizers using four and two phase protocols

Rostislav (Reuven) Dobkin and Ran Ginosar
VLSI Systems Research Center, Technion—Israel Institute of Technology, Haifa 32000, Israel
rostikd@tx.technion.ac.il
October 2007

Abstract

Synchronizers typically incur long latency of multiple clock cycles, resulting in low throughput. This work presents a number of novel fast synchronizers, based on four and two-phase protocols: a four and two-phase two-flop synchronizer which reduces the data cycle from 6-12 down to 2-4 clock cycles, and a four- and two-phase LDL synchronizer which strives for maximum throughput and "zero-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-flop synchronizers.

1. Introduction

Advanced silicon technologies enable the incorporation of an increasing number of modules in a single chip, constituting Systems on Chip (SoCs) devices. A SoC typically consists of multiple-clock domains, resulting from requirements for 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 dynamically changing clock relations during chip operation.

A SoC constructed of multiple modules, each working with uncorrelated local clock, is termed a Globally Asynchronous, Locally Synchronous (GALS) system [4][5]. Data synchronization and communication across clock domains in GALS architectures is a major challenge. Non-scaling interconnect complicates even more the synchronization problem. Increasing global wire delays incur high overhead, directly affecting communication performance. In addition, long wires have high variability in delay, both 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 mutual relationships of pairs of clock domains are classified in Table 1 according to the frequency and phase differences of the two domains. Mesochronous domains share the same frequency and have a constant phase difference between them, which can be compensated by relatively simple synchronizers [24][42], e.g. by a small FIFO. Adaptive phase compensation can be employed to connect multi-synchronous domains, in which the phase drifts slowly over time [40][41], as well as plesiochronous domains [43], where a very small frequency difference can be viewed as a phase drift. When two different-frequency clocks are used in the periodic case, a predictive synchronizer foresees and prevents contentions [44]. In the general asynchronous case, when the timing of input is unknown, the family of two-flop synchronizers and two-clock FIFOs can be employed. The synchronizers for the asynchronous class are universal because they also support all other classes. However, universal synchronizers do not take advantage of knowing the clock relationships and hence they are sub-optimal for other classes, incurring performance overhead. All the synchronizer types should be employed carefully, matching system requirements in terms of rate, latency and reliability, and avoiding common pitfalls [23].

Table 1: Clock relationship classes

Class	$\Delta\phi$	Δf	Synchronization
Synchronous	0	0	None
Mesochronous	ϕ_C	0	Phase compensation
Multi-synchronous	drifts	0	Adaptive phase compensation
Plesiochronous	varies	$\Delta f<\varepsilon$	Adaptive phase compensation
Periodic		$\Delta f>\varepsilon$	Predictive
Asynchronous			Two-Flop, two-clock FIFO

Simple "two-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 [22].

Two novel two-flop synchronizers are presented in this work. They employ four-phase and two-phase two-flip-flop synchronization circuits, and are shown to minimize latency and enhance throughput relative to conventional two-flop synchronizers.

The zero latency synchronizers proposed in this paper provide even higher throughput. They enable correct data sampling at the earliest possible edge of the sampling clock. The synchronizers are based on Locally Delayed Latching (LDL) [39] and requires no stopping of the clock. The proposed circuits employ standard interfaces, enabling seamless integration in modular SoC designs.

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 [40][41].

The paper is organized as follows. Sect. 2 compares low latency solutions, including a novel two-phase two-flop synchronizers, Sect. 3 presents zero latency LDL synchronizers, and simulations are described in Sect. 4.

2. Low latency synchronizers for asynchronous clock domains

2.1. Two-flop synchronizers

A. Four-phase synchronizer

Standard asynchronous "two-flop" synchronizers are widely employed in industry [23][24]. The main assumption of such synchronizer is that the time reserved for metastability resolution provides a satisfactory MTBF (mean time between failures). Latency of the simple synchronizer can be improved by employment of multiple sampling and either speculative or non-speculative voting [25][26].

A simple synchronizer is shown in Figure 1. The flip-flops sampling the asynchronous signals REQ and ACK may become metastable. One clock cycle is preserved for the metastability resolution, and no logic is allowed on the red lines. The exact time required for single synchronizer metastability resolution is derived from system MTBF requirement. When the time required for metastability resolution is longer than one clock cycle, additional flip-flops can be inserted before first flip-flop. Alternatively, when the requirement is shorter than one half clock cycle, a first falling edge flip-flop can be used. The receiver may also employ a FULL signal, pausing the synchronizer when the receiver is not ready to receive (ACK signal is not returned until FULL becomes low). In this work we consider a simple version, which assumes that the receiver is always ready. In this case the cross-lined flip-flops can be omitted, reducing the data cycle by two receiver clocks. Additionally, the data register at the RX side may also be omitted

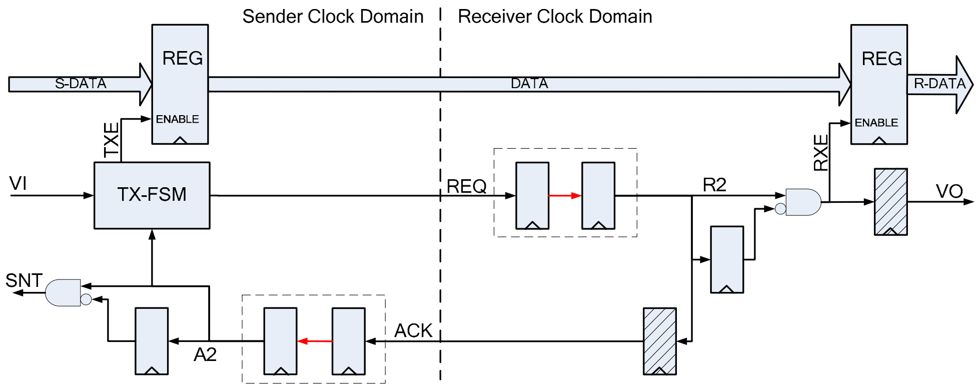


Figure 1: Simple four-phase synchronizer

The transmitter FSM and overall STG are shown in Figure 2. Note that '+' indicates a rising edge and '-' denotes a falling edge. The REQ is sent after input valid indication VI, provided that the synchronizer has finished its previous cycle (A2 is low). Output valid VO is pulsed for one RX cycle after a new data word has been received and synchronized, and sent indication SNT is pulsed for one TX cycle after A2 is received.

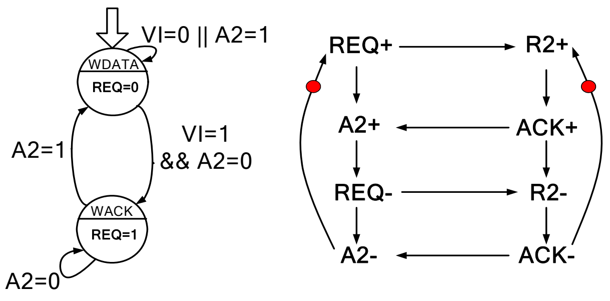


Figure 2. Sender FSM and synchronizer STG

The simple synchronizer enables reliable communication between two clock domains. Unfortunately, that synchronizer is limited to low data rates. In typical cases of mutually-asynchronous clocks, 6 TX cycles and 6 RX cycles are required for a complete and acknowledged transfer of a single word.

In Figure 3 an aggressive fast two-flop four-phase synchronizer is shown. The synchronization circuit in the receiver clock domain (right hand side) comprises F1 and the Enable input of REGR. F4 provides for acknowledgement, and F2, F3 generate a data valid single-cycle pulse. The time reserved for metastability resolution is one clock cycle, minus the setup time of the enable input to REGR. However, since the output of F1 branches to other targets, the resolution time is actually the minimum over all the (red) logic paths to F2, F3 and F4. The sender clock domain can be described similarly. Beware that the red lines require special treatment to allow for sufficient resolution time. They should not be combined with other parts of the logic. While other logic may be synthesized normally, caution should be applied to avoid manipulation of the red lines by the logic synthesizer and physical design software. In particular, note that certain registers have two separate enable inputs: one normal and one red, which cannot be simply merged by logic. 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 F1 or F5.

The operation of the synchronizer is explained by means of the TX FSM in Figure 4. At the beginning, the synchronizer waits for data (rising VI). The transmitter output registers (REGD and REGV) are enabled and will send out the new data word and REQ on the next rising edge of TX clock. At the receiver side, DATA is sampled by REGR and a VO pulse is generated. The timing depends on metastability resolution and may be delayed by an extra clock cycle. Note that metastability of the first sampling flip-flop can only result in non-determinism in timing, and R2 is not expected to assume an illegal voltage level (except, maybe, once per MTBF…). The receiver then produces the rising ACK signal. Once sampled, A2 disables TX output registers (REGD and REGV) and asynchronously resets REQ. The output registers stay disabled until the four-phase REQ/ACK handshake is over. The falling edge of R2 triggers an asynchronous de-assertion of ACK. Following the synchronized falling edge of A2, the transmitter enables the next data cycle once a new data word is available.

For a mesochronous operation, the minimal data cycle time (REQ+ → REQ+) is six clock cycles in the worst case (the two clocks are in phase), but only four clock cycles when the two clocks are out of phase (Figure 5). The synchronizer supports any relation between the transmitter and receiver clocks. When the clocks are mutually asynchronous then the data cycle depends largely on the slower clock. If the ratio is larger than two, then the data cycle is three clock cycles of the slower clock.

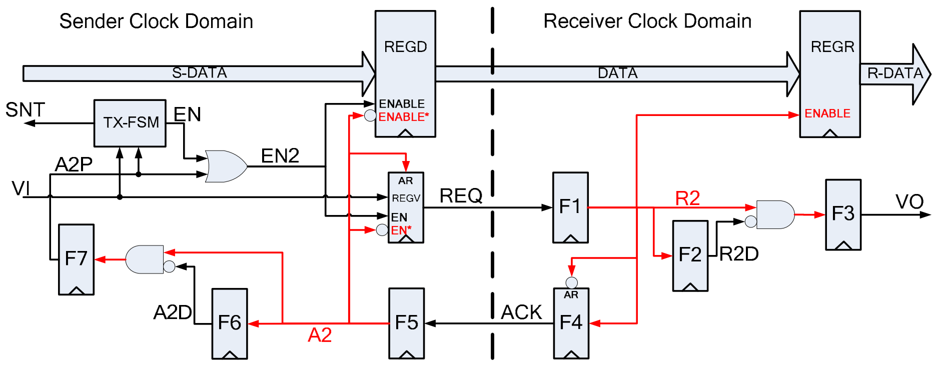


Figure 3. Fast two-flop four-phase synchronizer

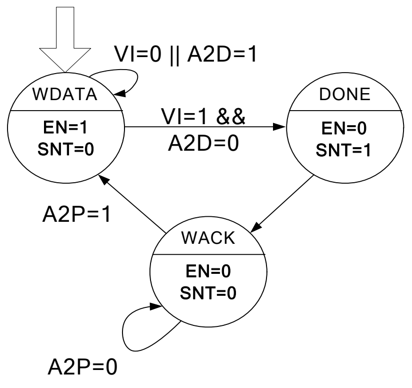


Figure 4. TX FSM of the fast four-phase synchronizer

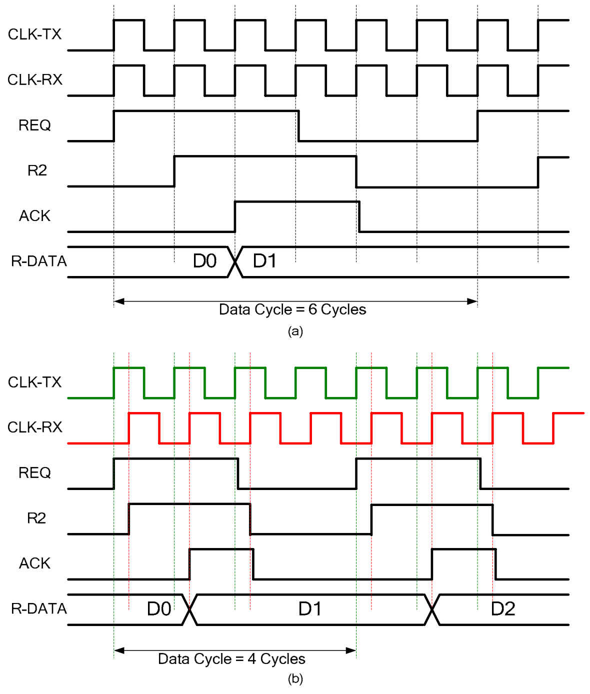


Figure 5: Mesochronous operation of the fast four-phase synchronizer: (a) in phase clocks (b) off-phase clocks.

B. Two-phase synchronizer

The main drawback of the synchronizer in Figure 3 is its high latency, which significantly affects data rate, preventing data transfer continuously on each clock cycle. The synchronization data-rate can be significantly improved by employing a two-phase protocol over the channel. This is particularly important for long range communication where wires incur additional high latency.

The fast two-phase synchronizer is shown in Figure 6. The synchronization circuit in the receiver clock domain (right hand side) comprises F1, the XOR gate and the Enable input of REGR. The XOR gate and the toggle F2 convert the two phase REQ into four-phase RXE and a single-cycle pulse VO. F4 provides for acknowledgement. The time reserved for metastability resolution is one clock cycle, minus the logic path delay from F1 the enable input of REGR. However, since the output of F1 branches to other targets, the resolution time is actually the minimum over all the (red) logic paths to F2, F3 and F4. The sender clock domain can be described similarly. Beware that the red lines require special treatment to allow for sufficient resolution time. They should not be combined with other parts of the logic. While other logic may be synthesized normally, caution should be applied to avoid manipulation of the red lines by the logic synthesizer and physical design software. In particular, note that certain registers have two separate enable inputs: one normal and one red, which cannot be simply merged by logic. 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 F1 and/or F5.

The synchronization circuit in the receiver clock domain (right hand side) comprises F1, the XOR gate and the Enable input of REGR. The XOR gate and the toggle F2 convert the two-phase REQ into four-phase RXE and a single-cycle pulse VO. F4 provides for acknowledgement. The time reserved for metastability resolution is one clock cycle, minus the logic path delay from F1 the enable input of REGR. However, since the output of F1 branches to other targets, the resolution time is actually the minimum over all the (red) logic paths to F2, F3 and F4. The sender clock domain can be described similarly. Beware that the red lines require special treatment to allow for sufficient resolution time. They should not be combined with other parts of the logic. While other logic may be synthesized normally, caution should be applied to avoid manipulation of the red lines by the logic synthesizer and physical design software. In particular, note that certain registers have two separate enable inputs: one normal and one red, which cannot be simply merged by logic. 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 F1 and/or F5.

The synchronizer operation is explained by the transmitter STG and FSM in Figure 7 and Figure 8 respectively. Note that TXS (the TX state) is derived from the (red) synchronization circuit and hence, its toggle time depends on metastability resolution, and can happen either one or two cycles after latching F5. 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 TXE (TX enable), the red-marked resolving signal from the sampling flip-flop.

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 (Figure 9). When the two clocks are mutually 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 (Figure 10). Table 2 summarizes the data cycle figures for all cases and for simple and fast four- and two-phase synchronizers.

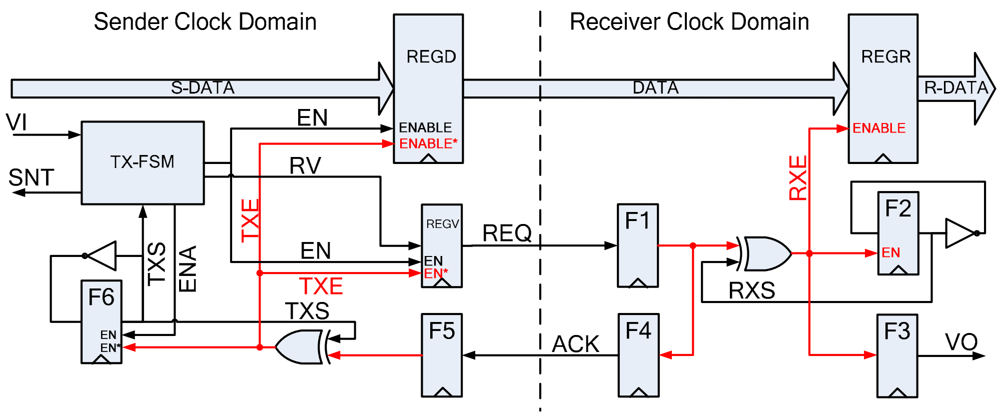


Figure 6: Fast two-flop two-phase synchronizer

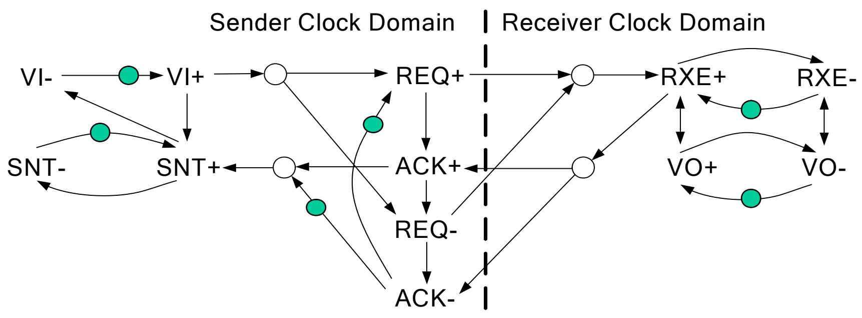


Figure 7. STG of the fast two-phase synchronizer

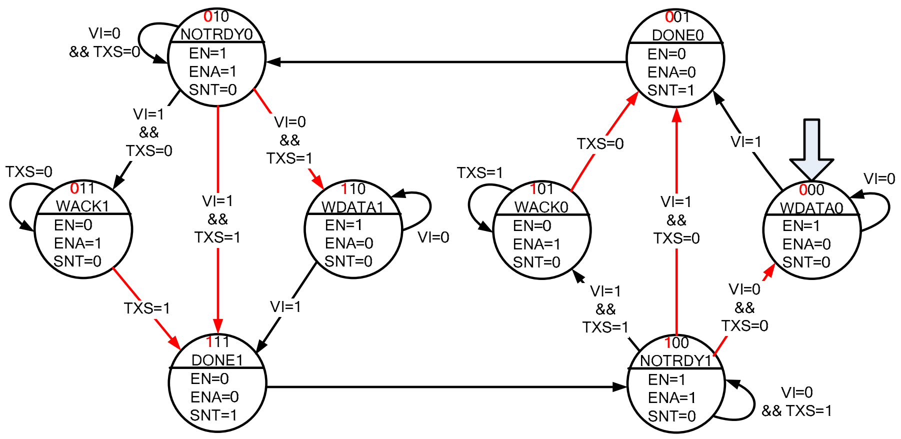

010 NOTRDY0 EN=1 ENA=1 SNT=0
001 DONE0 EN=0 ENA=0 SNT=1
011 WACK1 EN=0 ENA=1 SNT=0
110 WDATA1 EN=1 ENA=0 SNT=0
101 WACK0 EN=0 ENA=1 SNT=0
000 WDATA0 EN=1 ENA=0 SNT=0
111 DONE1 EN=0 ENA=0 SNT=1
100 NOTRDY1 EN=1 ENA=1 SNT=0
VI=0 && TXS=0
VI=1 && TXS=0
VI=0 && TXS=1
VI=1 && TXS=1
TXS=0
TXS=1
VI=0
VI=1
VI=1 && TXS=0
VI=0 && TXS=0
VI=1 && TXS=1
VI=0 && TXS=1

Figure 8: TX FSM of the fast two-phase synchronizer

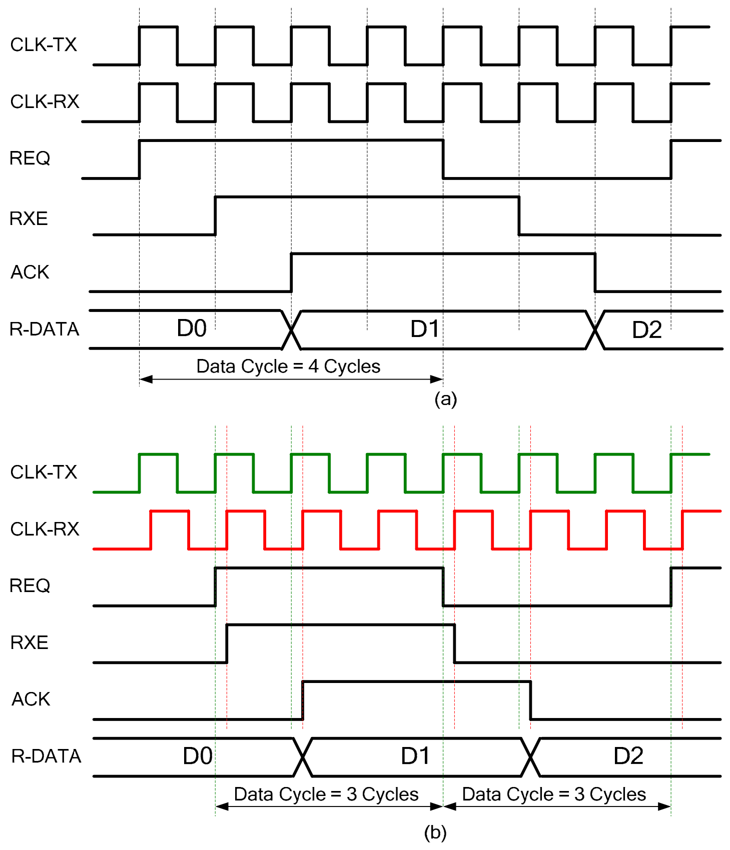


Figure 9: Mesochronous operation of the fast two-phase synchronizer: (a) clocks in phase (b) off-phase clocks

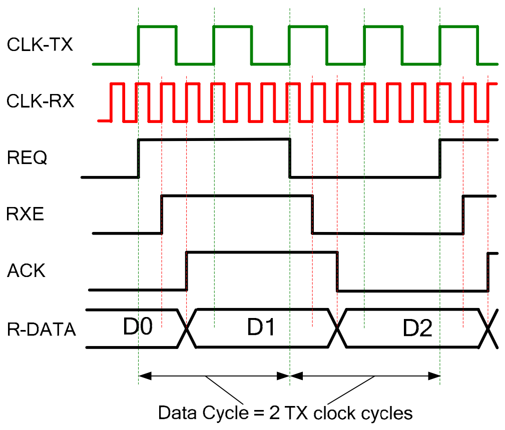


Figure 10: Asynchronous clock domains. One clock is three times faster leading to data cycle of two clock cycles of the slower clock

Table 2: Data cycles of two-flop synchronizers

	Simple Four-Phase		Fast Four-Phase		Fast Two-phase	
	Best (off-phase)	Worse (in-phase)	Best (off-phase)	Worse (in-phase)	Best (off-phase)	Worse (in-phase)
Mesochronous Clocks	10	12	4	6	3	4
Asynchronous Clocks	6·TX+6·RX		3·TX+3·RX		2·TX+2·RX	

2.2. Two-clock FIFO

The two-clock FIFO synchronizer enables transferring 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 communications over long interconnect. Long link delays affect both latency and data rate between two communicating modules. In [27], 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 [28] and expanded to multi-synchronous, plesiochronous and asynchronous cases in [29]. 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][30]-[32]. The technique incorporates a local ring-oscillator clock generator in each synchronous "island" with a set of MUTEXes [33] that stop the clock temporarily when new input data arrive. Handshake clocks [13] 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][30].

2.4. Categorizing synchronizer

Based on the works listed above, we categorize the synchronization approaches into a number of simple cases (Figure 11). 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.

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 Figure 11.

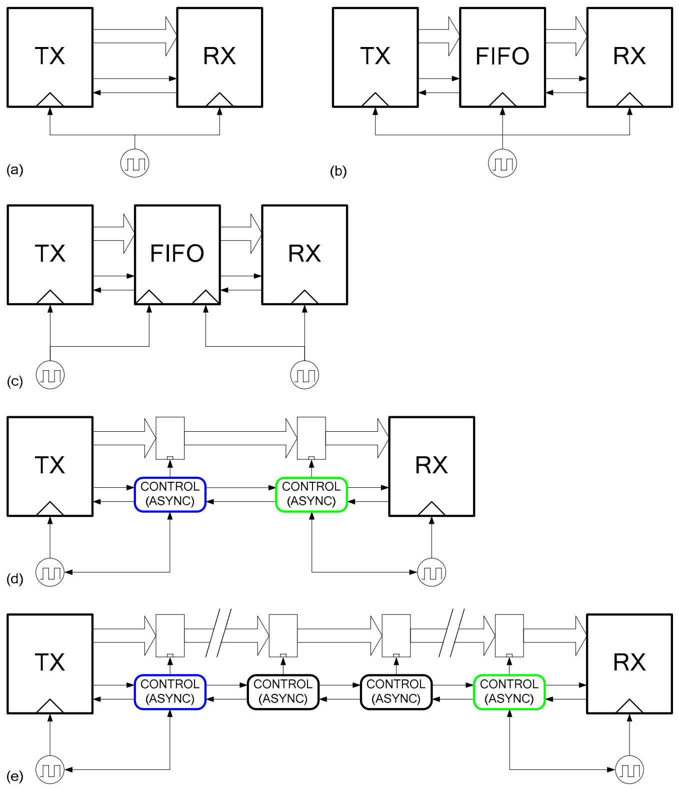


Figure 11: Fast synchronization: (a) single clock domain transfer, (b) buffered single clock domain transfer, (c) two-clock FIFO synchronizer, (d) stoppable clock and LDL synchronizers, (e) synchronization over long interconnect

3. Zero latency synchronizer

LDL synchronizer can achieve zero latency. The section starts with definitions, presents LDL concepts, and describes LDL input and output

3.1. Definitions

The *latency* of a synchronizer is defined as the time from the writing a data word into the output register of the sender (TX) to writing the same data into the first register of the receiver (RX). 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 the same goal in fast synchronizers whenever possible. Synchronizers that achieve that add no extra penalty due to synchronization to the basic latency of intra-clock domain transfers, and are hence called *zero latency synchronizers*. Figure 12 shows a timing example of a zero latency synchronizer: the TX data is sampled on the first RX clock following the transfer.

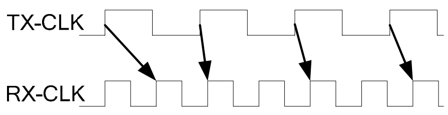


Figure 12: Zero latency synchronization

Interconnect delay affects both latency and throughput of the synchronizer. The latency is extended by the delay, while 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 delay latching [22] does not requires stopping the clock of locally synchronous islands. LDL is unaffected by any dynamic scaling of the clock cycle [1]-[3]. An asynchronous port (Figure 13) controls both the input latch and Y1, 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 and prevents write-after-read hazards.

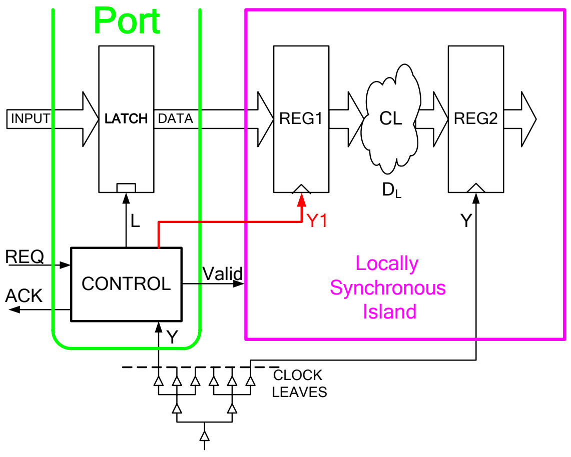


Figure 13: LDL input port connected to a synchronous island

Instead of stopping the clock Y, Y1+ is delayed when a conflict is imminent. Y1– is unaffected: only the high-phase (HP) is shortened. A port request is accepted only during the low-phase of Y, latching the incoming data (L+) and delaying Y1+ when needed. The conflicts between Y+ and REQ+ are resolved by a MUTEX inside the control. The time budgets of LDL synchronization are shown in Figure 14. In case of conflict (REQ+ and Y+), certain time is preserved for metastability resolution (M/S). Once metastability is resolved, the controller latches the incoming data (taking D_{CTRL} time). All this must complete at least HP time before Y–, so that the clock Y1 can be pulsed for at least HP time (the minimal pulse width required by the technology, usually 2-3 gate delays). 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. Note that M/S and HP are system requirements, while D_{CTRL} depends on the implementation of the controller.

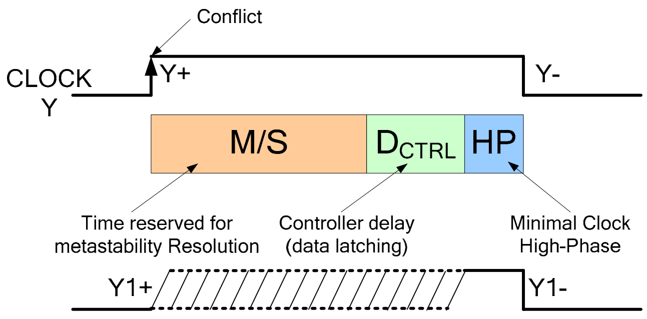

Conflict
CLOCK Y
Y+
Y-
M/S
D_{CTRL}
HP
Time reserved for metastability Resolution
Controller delay (data latching)
Minimal Clock High-Phase
Y1+
Y1-

Figure 14: LDL time budgets

The metastability resolution time requirement is derived from global SoC MTBF requirement. 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 [34]. This requirement can be achieved if the M/S period in Figure 14 is at least 43τ [22], where τ is the metastability resolution time constant and is assumed to be about one FO4 gate delay [35][33].

For T=100τ, this implies that almost one half of a symmetric clock cycle should be allowed for resolution. For slower SoCs, e.g. where the fastest clock cycle is 160τ, a quarter clock cycle suffices to achieve this MTBF. LDL can support faster clocks than T=100τ 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<50τ, a different approach based on multi-cycle resolution time or on multi-synchronous clocking is required.

The said three time intervals 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.

3.3. LDL input port

Figure 15 shows the LDL input port. An asynchronous controller provides for handshake with the external interface and for control of the MUTEX. In order to assure fastest operation, the external interface of the controller should be decoupled as much as possible from its internal interface. The ACK should be generated as soon as the data is latched inside REG, especially in the case of long range interconnect between TX and RX.

When the controller has data, it raises output ASK, which eventually leads LATCHED+ that captures the new data into REG. At that time, Y1 is kept low. We now compare the four- and two-phase versions of the asynchronous controller.

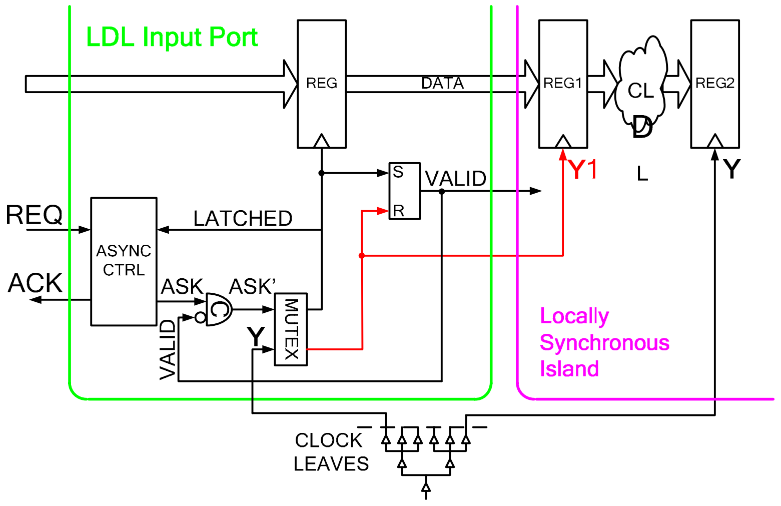


Figure 15: LDL input port

A. Four-phase input port

The STG specification of the asynchronous controller four-phase is shown in Figure 16a. The implementation of Figure 16b is simpler and faster than in [22]. The operation is exemplified in Figure 16c. D_{CTRL} comprises the latency LATCHED+ → ASK-→ASK'- (see Figure 15). Note that D_{CTRL} also depends on the load presented by REG, which depends on the data width.

The main drawback of this fast controller is the external interconnect delay penalty paid four times for each data transfer. When long-range interconnect is considered, this penalty is much higher than the internal delay of the controller.

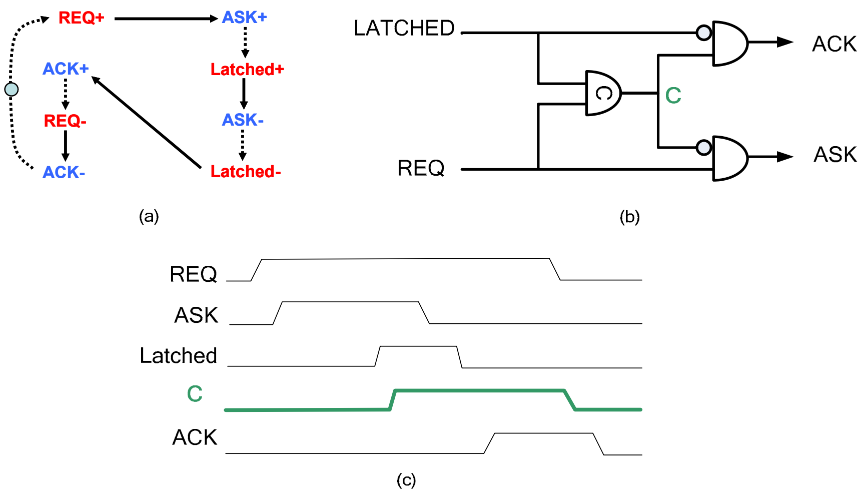


Figure 16: Four-phase input-port asynchronous controller

B. Two-phase input port

The two-phase protocol reduces the penalty of external interconnect delay by one half relative to the four-phase protocol. The two-phase controller STG, its circuit implementation and example waveforms are shown in Figure 17. This simple circuit employs only standard library cells, and it converts the two-phase protocol of REQ/ACK into the four-phase protocol of ASK/LATCHED.

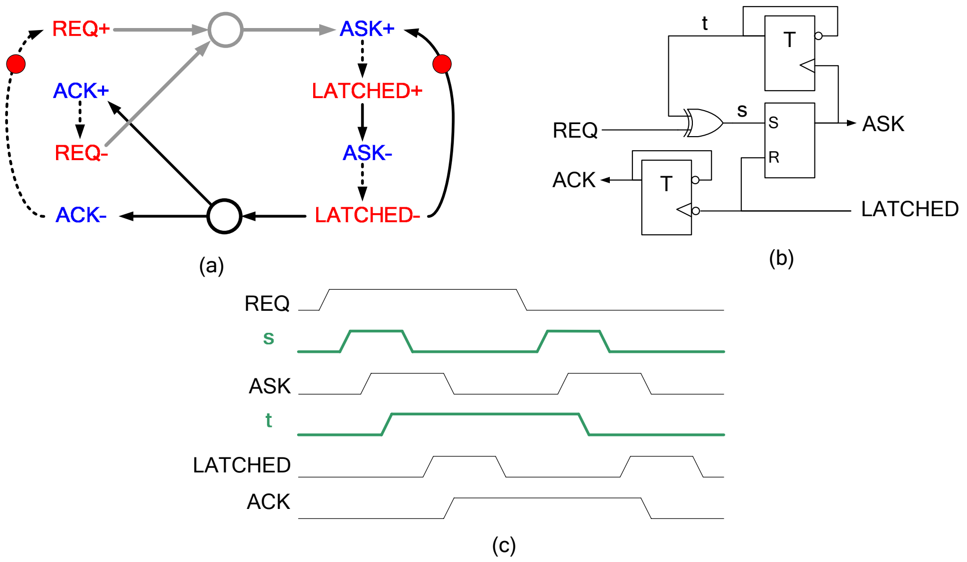


Figure 17: Two-phase input-port asynchronous controller

3.4. Output port

The output port is shown in Figure 18, in addition to special interface circuitry within the synchronous island. The output port interfaces the synchronous island by two signals, similarly to a standard FIFO handshake: VALID and FULL. Upon FULL, additional data are stalled. In addition, the output port generates the possibly delayed clock Y1.

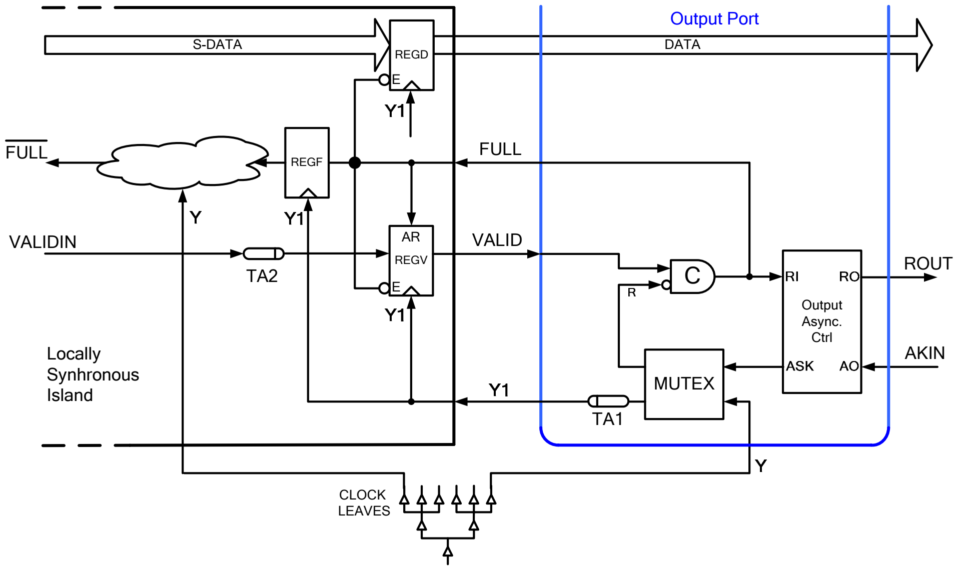


Figure 18: LDL output port

The output port must synchronize FULL, as specified in Figure 19: the assertion of VALID on Y1+ 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 Y1), thus preventing contention at the sampling register REGF.

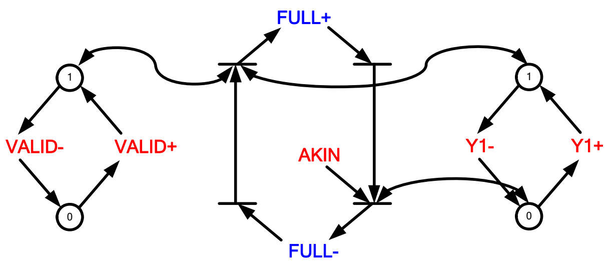


Figure 19: FULL signal generation

The overall transmitter operation is demonstrated in Figure 20. For each new data, VALID is asserted, leading to raising FULL and self-reseting VALID. 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 Figure 20). Thus, data can be transferred on each clock cycle of TX. When FULL is high during the rising edge of TX clock, the data is not changed (output flip-flops are disabled, case #2 in Figure 20). In the intermediate situation, when 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 Figure 20). In the second case, the acknowledge signal ASK wins over clock Y. Then, FULL is de-asserted and later on clock Y1 is unblocked, resulting in shortened Y1 cycle (case #4 in Figure 20). Note that in both #3 and #4 cases we have to retain the next data over the normal clock edge, which is obtained by connecting DATA and VALID registers to clock Y1. In the following we discuss implementation issues of the output port and its interface.

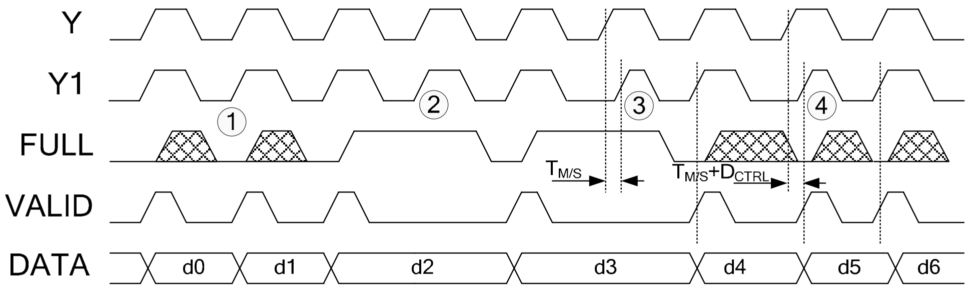


Figure 20: LDL output port operation

A. Timing assumptions

The LDL output port of Figure 18 requires two timing assumptions as follows.

TA1: Whereas both E+ (the enable of REGD, REGV) and Y1+ emanate from FULL−, the former must precede the latter (Eq. (1)). This requirement is easily met in the circuit (**TA1**=0 in Figure 18).

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

$$\begin{tikzcd} & E+ \arrow[dr, "T_{SU}"] & \\ FULL- \arrow[ur] \arrow[r] & ASK- \arrow[r] & Y1+ \end{tikzcd}$$

Figure 21: Timing assumption #1

TA2: Hold time should be satisfied for registers REGD, 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 Figure 20). 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:

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

The value of $T_{MUTEX}+D_{CTRL}$ is very high and therefore hard to meet by simple delay line. Below we present a possible implementations of the interface that supports the requirement of (2).

B. Four-phase output port

The four-phase controller specification is shown in Figure 22a. The straightforward implementation by Petrify [36] leads to a relatively complex and slow circuit (Figure 22b), which operation is shown in Figure 22c. Along with the internal latency overhead, the main drawback of this controller is the penalty of external interconnect delay paid four times per each data transfer.

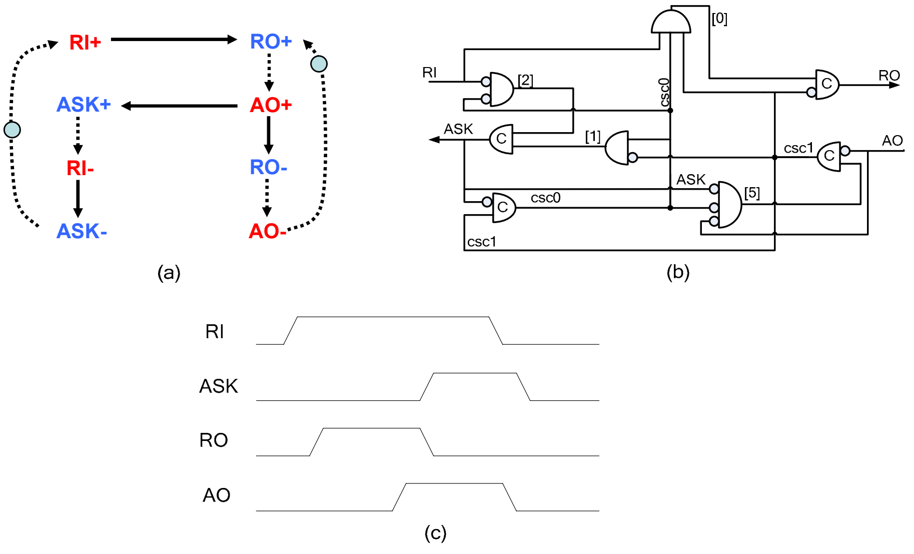


Figure 22: Four-phase output-port asynchronous controller

C. Two-phase output port

The STG of the asynchronous controller of the output port, its circuit and example waveforms are shown in Figure 23. 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 Figure 18) 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.

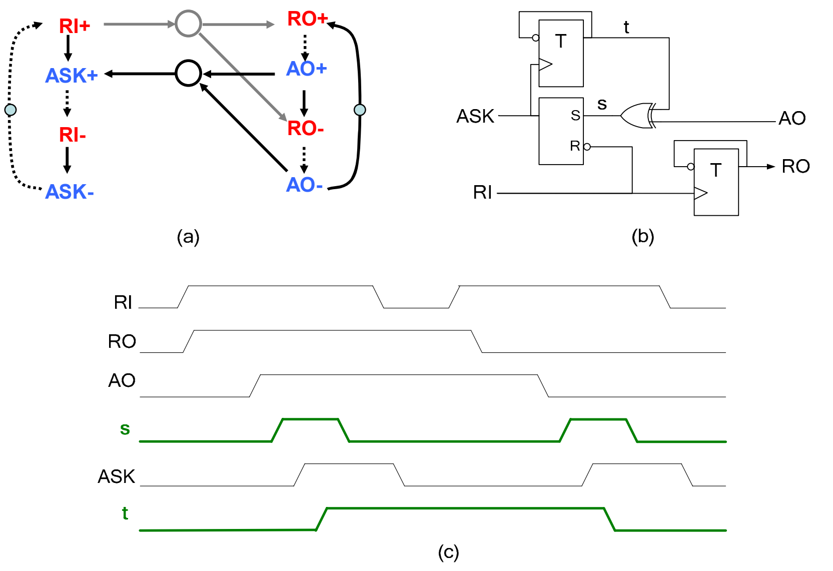


Figure 23: Two-phase output-port asynchronous controller

D. Synchronous interface to the output port

The zero latency LDL synchronizer supports throughput of one DPC (data item per cycle). To enable that, the synchronous interface must be able to supply data words at the highest rate, while satisfying the timing assumptions above. In this section we show an implementation example that can be used either as an additional stage of the synchronous module or can be integrated into a special, synchronous FIFO that is connected to the LDL output port.

SDATA (Figure 18) 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 clock Y. To avoid loss of the word currently on SDATA, an auxiliary registers [37] RAUX, RAUXV are added (Figure 24). The hold requirement (2) is satisfied by the negative edge triggered PRE_REGD and PRE_REGV registers, which stabilize the inputs of REGD, REGV during next Y1+, even when Y1 is skewed 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.

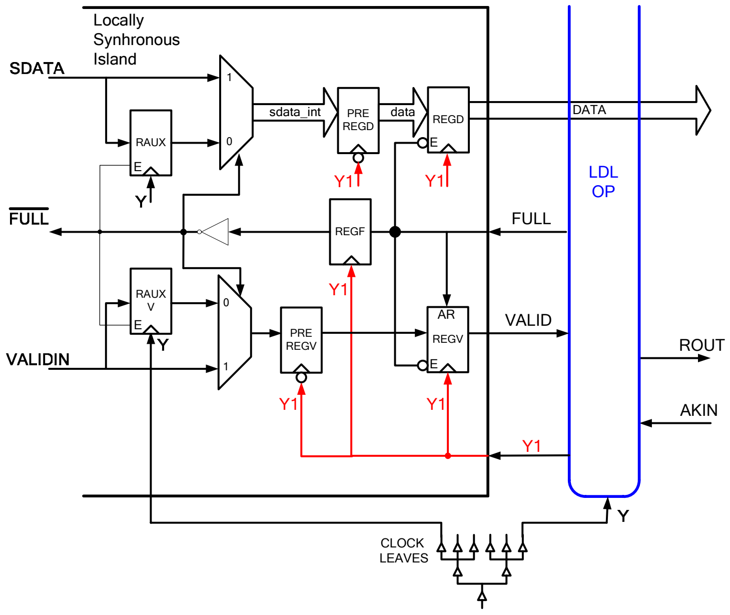
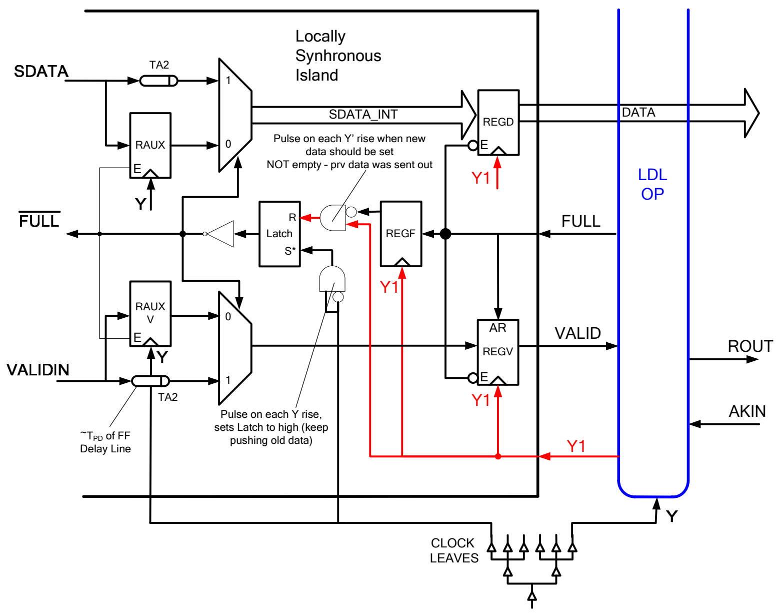


Figure 24: Synchronous output interface – detailed scheme

Another approach to fulfill the hold requirement (2) is shown in Figure 25. In this circuit, after each clock Y rise, SDATA_INT is switched to the value that is stored inside RAUX registers, holding the old value on the SDATA bus until FULL_INT indication is updated during the corresponding Y1 rise. In case FULL_INT equals to logic high (FULL signal was sampled) the SDATA bus stays connected to the RAUX register, preserving the non-sent data word at the REGD input. In the other case, when FULL_INT equals to logic zero (the data that was previously on SDATA_INT bus was sampled into REGD), the latch L is reset and SDATA_INT is connected to SDATA bus, preparing a new data value at the input of REGD. The pulse generators outputs used in Figure 25 are slightly delayed relative to each other in order to prevent possible concurrent rise.



Locally Synhronous Island
SDATA
TA2
RAUX
E
Y
1
0
SDATA_INT
Pulse on each Y' rise when new data should be set NOT empty - prv data was sent out
REGD
E
Y1
DATA
LDL OP
FULL
R Latch S*
REGF
Y1
FULL
RAUX V
E
Y
VALIDIN
TA2
~T_PD of FF Delay Line
Pulse on each Y rise, sets Latch to high (keep pushing old data)
AR REGV
E
Y1
VALID
ROUT
AKIN
Y1
Y
CLOCK LEAVES

Figure 25: Output port interface. Small hold requirement on the DATA and VALID wires (of a few gate delays)

4. Performance results and analysis

In this section we compare the performance of the zero latency LDL synchronizer with the fast two-phase two-flop synchronizer of Sect. 2 and with standard two-clock FIFO synchronizer. We use a standard FIFO from Synopsys DesignWare library [38], setting internal synchronizer depths to two. A FIFO depth of 10 and bursts of 1,000 words were used in the simulation.

4.1. Latency

Figure 26(a) shows upper and lower bounds of the LDL synchronizer latency for back-to-back connection (no interconnect delay). The latency is lower than half a cycle when RX clock is faster than TX, clearly indicating zero 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 clocks result in increased latency difference between best and worst cases, as well as other special values of the ratio of cycle times (see Sect. 4.3) .

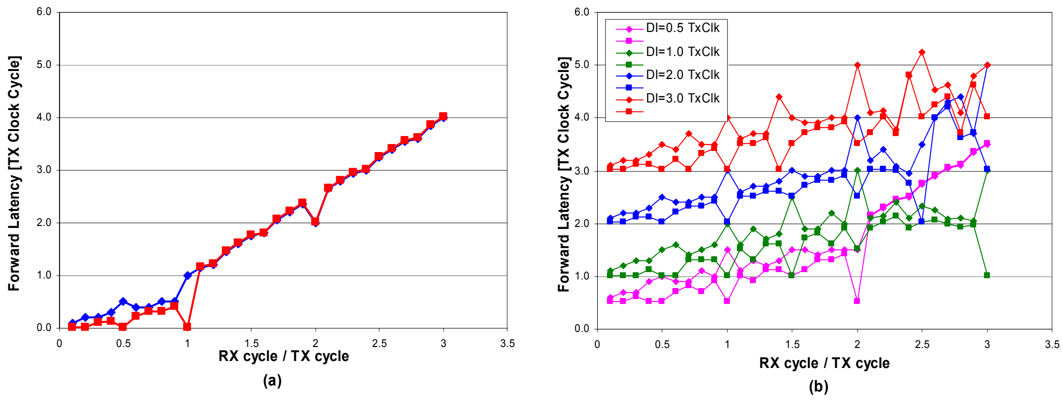


Figure 26: LDL synchronization latency bounds

Figure 27 shows that the worst case latency of the FIFO is the longest, and the LDL synchronizer incurs the least latency. The two-flop synchronizer latency lies in-between.

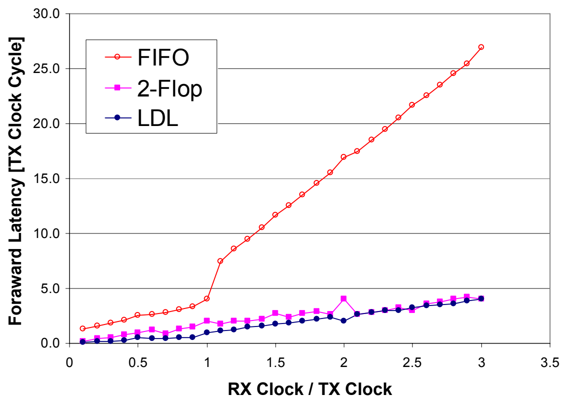


Figure 27: Latency of LDL, 2-flop and 2-Clock FIFO synchronizers

The LDL synchronizer outperforms the two-flop synchronizers even when the interconnect delays are considered (Figure 28). The simulations show latency reduction down to a factor of three. 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.

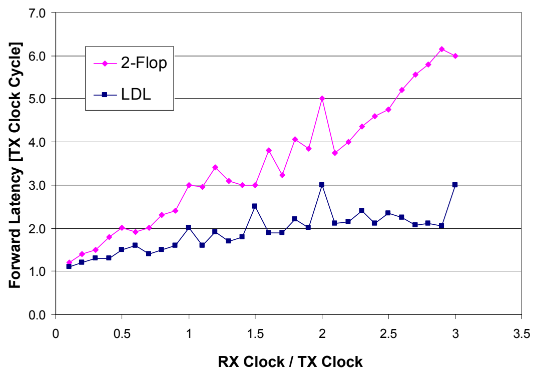


Figure 28: Latency of LDL and 2-Flop synchronizers for 1.0·TX-CLK interconnect delay

4.2. Data Rate

Bounds of data rate of the LDL synchronizer are shown in Figure 29 for different interconnect delays. Note that twice the interconnect delay, namely the flight time of REQ and ACK, is the theoretical lower bound of the data cycle. 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 Figure 26.

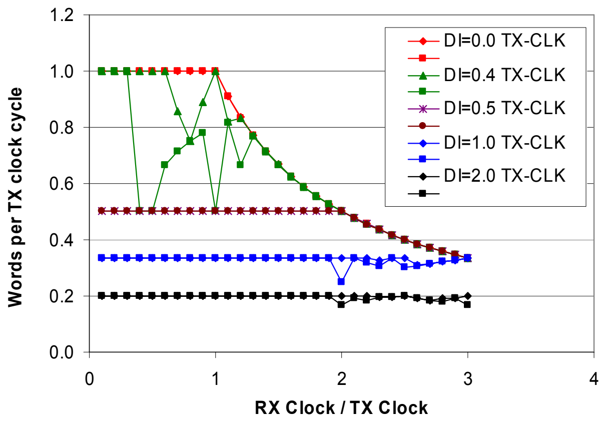


Figure 29: LDL synchronization throughput bounds

LDL and FIFO throughputs are similar as evident from the overlapping charts in Figure 30 and are about twice faster than the fast two-flop two-phase synchronizer (as expected, see Table 2). 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-flop synchronizer.

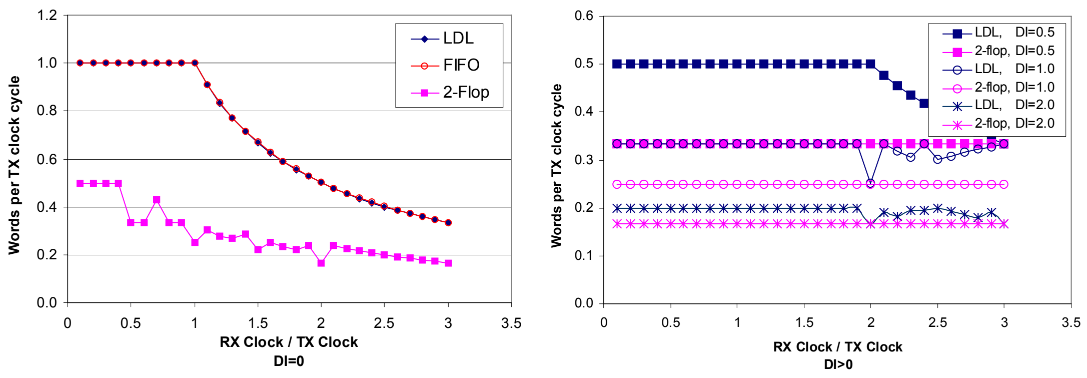


Figure 30: Throughput comparison for LDL, 2-Flop and FIFO synchronizers

4.3. Data transfer and clock relation taxonomy

Certain clock relationships may bring synchronizers to work in best or worst case regions of their performance (see the upper and lower bounds in Figure 26 and Figure 29). In this section we categorize the relationship and discuss the different performance cases.

We distinguish two types of data transfer that lead to different synchronizer performance: single data word transfer, and the transfer of data burst (Figure 31). In case of single word transfer, the best case (BC) latency is D_I, the latency of interconnect. In the worst case (WC), the data word just misses the rising edge of RX clock and is detained for an additional RX clock cycle, T_{RX}. This is also exemplified in Figure 32.

$$L_{SINGLE\ WORD} = \begin{cases} D_I + T_{RX}, & WC \\ D_I, & BC \end{cases} \tag{3}$$

On burst transfer, the synchronization of one word may affect the synchronization of the next one. Clock relations also affect the performance. We distinguish informally two cases, “short” and “long” locks. Long locked clocks conflict rarely, either because they have similar frequencies or one is significantly slower than the other. As a result, the relative phase of the two clocks drifts (namely changes slowly, or changes very little from one cycle to the next). Short locked clocks, on the other hand, demonstrate a short period between successive clock conflicts.

When there is a short periodic relation between the clocks ("short-lock"), then a certain number of same relation cases appear periodically. In this case the data transfer can be held stable in a specific case. In some cases, due to delays, the short-lock cannot be locked at the WC/BC cases and moves to an average point between them. In Figure 33 an example of stable and unstable cases is shown. Two-phase REQ/ACK protocol is assumed. Data is latched at the receiver side during low phase of RX-CLK upon a pending request REQ. ACK is generated asynchronously for each new data latching. Stable case (Figure 33a) is shown for back-to-back connection, D_I=0, and clock ratio of two. In this case the latency is $2 \cdot T_{TX}$ and it is equal to the WC latency of single word synchronization (Eq. (3)). When an interconnect delay is incurred ($D_I = 0.5 \cdot T_{TX}$ in Figure 33b), the WC latency for a single word synchronization is $2.5 \cdot T_{TX}$ (Eq. (3)). However, the WC latency is paid only for the first cycle of the burst, while for the subsequent cycles, thanks to the forward and backward interconnect delays, the latency converges to another value of $1.5 \cdot T_{TX}$. Note that first cycle acknowledge ACK is blocked since it arrives at TX after TX-CLK rise and therefore it is served only on the next TX-CLK cycle.

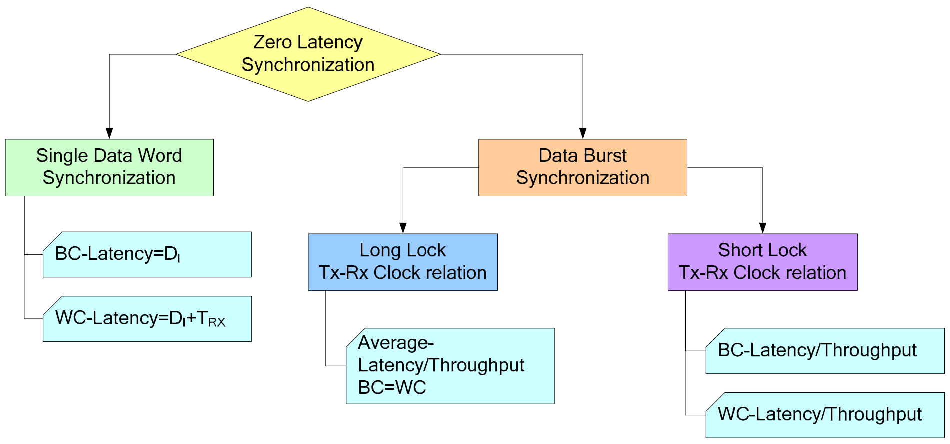

Zero Latency Synchronization
Single Data Word Synchronization
Data Burst Synchronization
BC-Latency=D_l
WC-Latency=D_l+T_{RX}
Long Lock Tx-Rx Clock relation
Short Lock Tx-Rx Clock relation
Average-Latency/Throughput BC=WC
BC-Latency/Throughput
WC-Latency/Throughput

Figure 31: Synchronization Best- and Worst-cases for different data patterns and clock relations

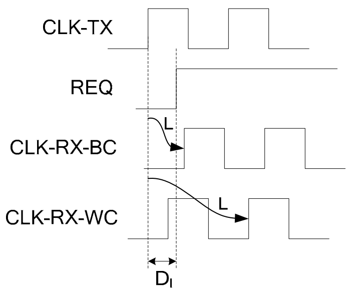


Figure 32: Best and Worst case request arrivals

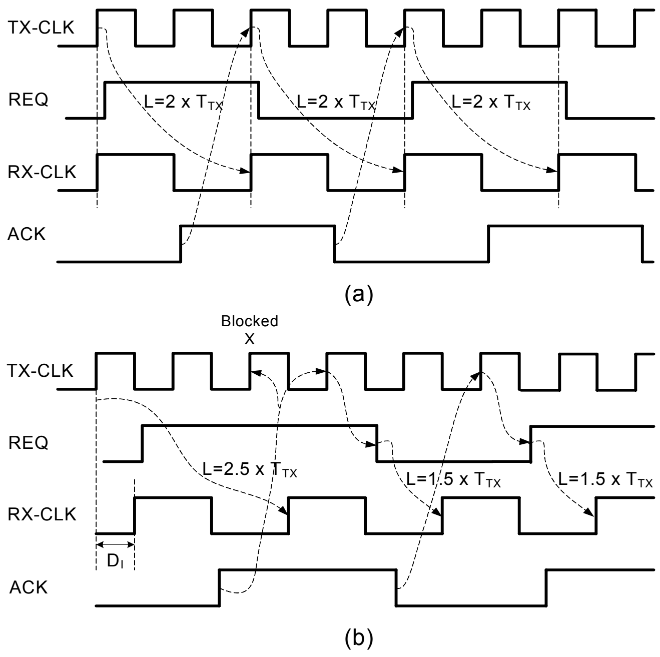


Figure 33: (a) Stable and (b) unstable worst cases for burst synchronization. REQ and ACK are shown for RX side

5. Conclusions

Aggressive novel synchronizers that employ four- and two-phase protocols have been presented. The synchronizers are based on a two-flop and a LDL families. They facilitate clock domain crossings both when the two domains are physically adjacent and when they are separated by long interconnect.

The two-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-flop synchronizer. The LDL synchronizer is termed ‘zero latency’ because it does not add any latency penalty when crossing clock domains, and it enables sending data every clock cycle. The LDL zero 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 2-flop) in 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," Proc. of IEEE/ACM International Symposium on Microarchitecture, pp. 356-367, 2002.

[2] L. S. Nielsen, C. Niessen, J. Sparsø, and C. H. van Berkel. Low-power operation using self-timed and adaptive scaling of the supply voltage. IEEE Transactions on VLSI Systems, 2(4):391-397, December 1994.

[3] W.R. Daasch, C.H. Lim, G. Cai, "Design of VLSI CMOS Circuits Under Thermal Constraint," IEEE Transactions on VLSI Systems, 49(8), 589-593, Aug. 2002.

[4] D.M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," PhD Dissertation, Stanford University, 1984.

[5] D. Bormann, P. Cheung, "Asynchronous Wrapper for Heterogeneous Systems," Proc. of ICCD, pp.:307 - 314, 1997.

[6] L. Scheffer, "An Overview of On-chip Interconnect Variation," Proc. SLIP, pp. 27-28, 2006

[7] R. O. Topaloglu, A. B. Kahng, "Generation of Design Guarantees for Interconnect Matching," Proc. SLIP, pp. 29-34, 2006.

[8] J.Bainbridge and S.Furber, "Chain: a Delay-Insensitive Chip Area Interconnect," IEEE Micro, 22(5):16-23, 2002.

[9] E. Beigne, F.Clermidy, P.Vivet, A.Clouard, M. Renaudin, "An Asynchronous NOC Architecture Providing Low Latency Service and Multi-Level Design Framework," Proc. of ASYNC, 54-63, 2005.

[10] T Felicijan, S.B. Furber, "An Asynchronous On-Chip Network Router with Quality-of-Service (QoS) Support," Proc. of IEEE Int. SOC Conf., 274-277, 2004.

[11] T. Bjerregaard, J. Sparso, "A Scheduling discipline for latency and Bandwidth Guarantees in Asynchronous Network-on-chip", Proc. ASYNC, 34-43, 2005.

[12] R. Dobkin, V. Vishnyakov, E. Friedman and R. Ginosar, "An Asynchronous Router for Multiple Service Levels Networks on Chip," Proc. ASYNC, 44-53, 2005.

[13] International Technology Roadmap for Semiconductors (ITRS), 2005, www.itrs.net

[14] J. Kessels, A. Peeters, P. Wielage, and S.J. Kim, "Clock Synchronization through Handshake Signaling," Proc. of ASYNC’02, pp. 59-68, 2002.

[15] S. Moore, G. Taylor, R. Mullins, P. Robinson, "Point to Point GALS Interconnect," Proc. of ASYNC'02, pp. 69-75, 2002.

[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," Proc. of ASYNC’03, pp. 141-150, 2003.

[18] J. Muttersbach, T. Villiger, W. Fichtner, "Practical Design of Globally-Asynchronous Locally-Synchronous Systems," Proc. of ASYNC’00, pp. 52-61, 2000.

[19] K. Y. Yun, R.P. Donohue, "Pausible clocking: a first step toward heterogeneous systems," Proc. of Computer Design: VLSI in Computers and Processors, ICCD'96, pp. 118–123, 1996.

[20] K. Y. Yun, R.P. Donohue, "Pausible clocking-based heterogeneous systems," IEEE Transactions on VLSI Systems, 7(4), pp. 482-488, 1999.

[21] A. E. Sjogren and C. J. Myers, "Interfacing Synchronous and Asynchronous Modules within a High-Speed Pipeline," IEEE Transactions on VLSI Systems, 8(5), pp. 573–583, 2000.

[22] R. Dobkin, R. Ginosar and C. P. Sotiriou, "High Rate Data Synchronization in GALS SoCs," TVLSI, 14(10):1063-1074, 2006.

[23] R. Ginosar, "Fourteen Ways to Fool Your Synchronizer," ASYNC, 89-96, 2003.

[24] W.J. Dally, J.W. Poulton, Digital Systems Engineering, Cambridge university press, Cambridge, UK, 1998.

[25] D.J. Kinniment, A. Yakovlev, "Low latency synchronization through speculation," PATMOS, 278-288, 2004.

[26] S.J. Kim, J.G. Lee, K. Kim, "A parallel flop synchronizer for bridging asynchronous clock domains," AP-ASIC, 184-187, 2004.

[27] T. Chelcea, S. M. Nowick, "Robust interfaces for mixed-timing systems," IEEE Transactions on VLSI, 12(8), 857-873, 2004.

[28] A. Chakraborty, M. R. Greenstreet, "Efficient self-timed interfaces for crossing clock domains," Proc. of ASYNC 2003, pp. 78-88, 2003.

[29] A. Chakraborty, M. R. Greenstreet, "A minimal source-synchronous interface," Proc. ASIC/SOC Conference, pp.443 – 447, 2002.

[30] S. Chakraborty, Joycee Mekie, and D.K. Sharma, "Reasoning about Synchronization Techniques in GALS

Systems: A Unified Approach," Proc. Workshop on Formal Methods in GALS Architectures (FMGALS), 2003.
[31] J. Mekie, S. Chakraborty, D.K. Sharma, "Evaluation of pausible clocking for interfacing high speed IP cores in GALS framework," Proc. of VLSI Design, pp. 559-564, 2004.
[32] R. Mullins, S. Moore, "Demystifying Data-Driven and Pausible Clocking Schemes," Proc. of ASYNC'07, pp. 175-185, 2007.
[33] C.L. Seitz, "System timing," in C.A. Mead and L.A. Conway, eds., Introduction to VLSI Systems, ch. 7, Addison-Wesley, 1980.
[34] R. Ginosar, "MTBF of multi-synchronizer SoC," http://www.ee.technion.ac.il/~ran/papers/MTBFmultiSyncSoc.pdf.
[35] C. Dike and E. Burton, "Miller and Noise Effects in a Synchronizing Flip-flop," IEEE Journal of Solid-State Circuits, 34(6), pp. 849-855, 1999.
[36] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," IEICE Transactions on Information and Systems, Vol. E80- D, No. 3, 315– 325, 1997.
[37] L. Carloni, A. Sangiovanni-Vincentelli, "Coping with latency in SoC design," IEEE Micro, special issue on SoC, 22(5):24- 35, 2002.
[38] Synopsys Design Ware FIFO, www.synopsys.com/products/designware/docs/doc/dwf/datasheets/dw_fifo_s2_sf.pdf.
[39] R. Dobkin, R. Ginosar, C. Sotiriou, "Data Synchronization Issues in GALS SoCs," ASYNC, 170-179, 2004.
[40] Y. Semiat, R.Ginosar, "Timing Measurements of Synchronization Circuits," ASYNC, 68-77, 2003.
[41] R. Kol, R. Ginosar, "Adaptive Synchronization", ICCD, 188-189, 1998.
[42] T. H.-Y. Meng, Synchronization Design for Digital Systems(Eds.): Kluwer Academic Publishers, 1991.
[43] L.R. Dennison, W.J. Dally, D. Xanthopoulos, "Low-latency plesiochronous data retiming," Advanced Research in VLSI, pp. 304–315, 1995.
[44] U. Frank, T. Kapschitz and R. Ginosar, "A predictive synchronizer for periodic clock domains," J. Formal Methods in System Design, 28(2):171-186, 2006.