Reasoning about Synchronization Techniques in GALS Systems: A Unified Approach

Supratik Chakraborty, Joycee Mekie, and Dinesh K. Sharma

Indian Institute of Technology, Bombay
Powai, Mumbai 400076, INDIA

Abstract. Correct design of interface circuits is crucial for the development of high-speed SoCs (GALS systems) using IP cores. In this paper, we propose abstract timing diagrams as a formalism for reasoning about synchronization issues in different interface designs at a high level of abstraction. The advantage of using this formalism is that it permits reasoning about synchronization problems in interfacing schemes, even without knowing low-level implementation details. We show that it is possible to reason about the correctness and limitations of several existing interfacing schemes using this formalism. We also provide an example where reasoning using abstract timing diagrams guides us towards designing a new interface circuit for high-speed SoCs.

1 Introduction

Recent advances in VLSI technology have made possible the integration of complex subsystems like microprocessors, digital signal processors, embedded controllers, etc. on a single die. As technology matures further, we are fast approaching an era where entire systems consisting of multiple components can be etched on the same die, giving rise to Systems-on-Chip (SoC). While some SoCs have already seen the light of the day, chips with significantly higher complexity and with multiple clock domains are foreseeable in the near future. Globally Asynchronous Locally Synchronous (GALS) architectures offer a promising approach to manage the design complexity of such systems. Specifically, GALS architectures are suitable for systems-on-chip designed using IP cores. Individual cores like microprocessors and digitial signal processors can be viewed as *locally synchronous* modules (henceforth called *LS* modules) that communicate asynchronously to implement the overall functionality of the system. An SoC designer using IP cores is thus relieved of the burden of designing each module down to its last transistor. Her primary responsibility now includes designing interfaces between modules to ensure error-free communication without adversely affecting the overall performance of the system. *Correct interface design is therefore a crucial activity for sucessful development of SoCs of the future.* In this paper, we focus on interface designs in GALS systems exemplified by SoCs with multiple IP cores. Thus, in the remainder of our discussion, we will use the terms *SoC* and *GALS system* interchangeably. Similarly, we will use the terms *LS module*, *IP core* and *core* interchangeably.

While interface design is not a new discipline, several new issues arise when interfacing high speed IP cores in large SoCs. For example, since the internals of an IP core cannot be modified, an interface designer cannot enforce arbitrary constraints between signals going in and coming out of a core. The controllability and observability of signals within cores are fundamentally limited by the input/output signals of the core. Similarly, if individual modules are clocked at very high frequencies, and if inter-module interconnect delays are large and have transient variations compared to the clock period, communication between modules can no longer be treated as synchronous. This necessitates the development of asynchronous communication protocols and high-speed synchronization techniques at the boundary of each LS module. The effect of these protocols and synchronization techniques on the overall performance of the system, and their robustness in interfacing high-speed modules with unpredictable communication patterns are important issues that require detailed exploration. On another dimension, the presence of large clock buffering trees within high-speed IP cores can cause interfacing schemes, like pausible clocking [1], that work well at lower frequencies to malfunction at higher frequencies of operation. Thus, the limits of each interfacing technique must be carefully analyzed before it can be applied in a specific context. If different subsystems within an SoC are fed by independent clocks without pre-determined phase and frequency relationships, the interfacing problem becomes even more complex.

It is therefore clear that designing correct interfaces between IP cores in high-speed SoCs is a challenging problem, and presents research opportunities along several dimensions. To name a few, we need innovative analog and digital circuit designs, timing analysis tools at multiple levels of abstraction, techniques for the design of communication protocols and synchronization strategies that work correctly at high speeds of operation, formal verification of such protocols and synchronization techniques, tools for failure and metastability analysis, and performance analysis at both component and system levels. Our goal in this paper is to address one of these issues. Specifically, we provide a uniform framework for reasoning about the correctness of signal timing in different interfacing schemes used in GALS systems. We show that it is possible to use a simple formalism to reason about the sychronization problem at a high level of abstraction. We also show that it is possible to refine our analysis if details of a specific interfacing scheme are available. This allows us to study interfacing schemes without necessarily implementing them. Such an analysis also leads to discovery of constraints under which an interfacing scheme is able to achieve error-free communication between LS modules. The discovery of such constraints points to the infeasibility of certain interfacing schemes, and guides us in improving existing solutions and developing new interfaces.

The literature contains an impressive collection of interfacing schemes for GALS systems [2],[1],[3],[4],[5]. While some schemes, like those based on synchronizers [6], are applicable in a general setting, they suffer from a non-zero probability of failure. Other schemes like STARI [7], adaptive synchronization [8] or pausible clocking [1],[9] guarantee error-free communication under assumptions about delays, and frequency and/or phase relations between the clocks of communicating modules. Each proposal in the literature is accompanied by a detailed analysis indicating the conditions under which error-free communication can be guaranteed, and the probability of failure, if any. Unfortunately, different approaches use different analysis techniques, and it is often difficult to compare them in the same framework. The framework proposed in this paper allows us to analyze the synchronization problem in various interfacing schemes in a uniform manner. This enables us to see their comparative strengths and weaknesses.

We propose a timing diagram representation that allows us to reason about synchronization issues in GALS systems visually. While similar representations have been used earlier in other contexts [10],[11],[12], we show that our representation offers visual cues about why certain interfacing schemes work under restricted conditions but fail otherwise. For example, our diagrams clearly indicate that existing pausible clocking schemes cannot guarantee freedom from metastability-induced errors when clock buffering delays are large compared to the clock period, and the rate of communication is high. Interestingly, we also obain a constraint between the rate of communication, clock frequency and intra-module clock buffering delay which, if satisfied, allows existing pausible clocking schemes to be extended in a simple manner to systems with large clock-buffering delays. Our analysis also indicates that the interfacing problem can be solved robustly if we are able to design certain new kinds of circuits.

The remainder of this paper is organized as follows. Section 2 describes abstract timing diagrams as a formalism for describing and reasoning about synchronization issues in GALS interfaces at a high level of abstraction. We show how generic constraints for correct operation of interfaces can be derived from these diagrams. Section 3 describes the use of abstract timing diagrams in reasoning about the correctness of specific interfacing schemes. We also describe how limitations of individual schemes can be derived using abstract timing diagram based reasoning. Section 4 gives an example of how abstract timing diagrams can be used to identify features of circuit elements that can help in developing robust interfacing solutions. Specifically, we show that if a mutual exclusion element could be designed such that it always resolves metastability in one direction (possibly after an unbounded time), or if there exists a *lower* time bound for the mutex element to resolve metastability (in either direction), we could interface independent clock domains robustly. Finally, section 5 summarizes the role of abstract timing diagrams in reasoning about synchronization issues in GALS systems, and points to future directions of research.

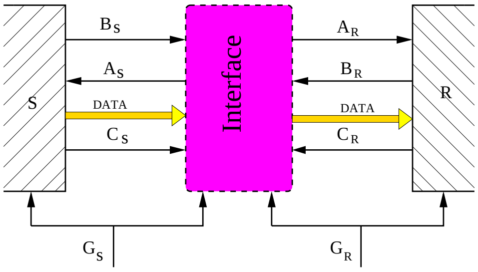


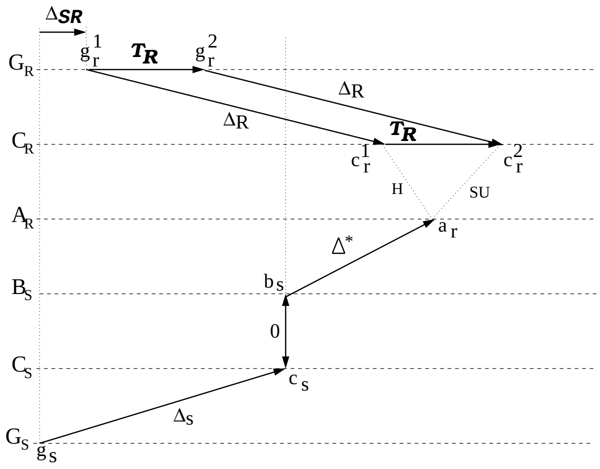
Fig. 1. Abstraction of interface between LS modules

2 Abstract timing diagrams for GALS interfaces

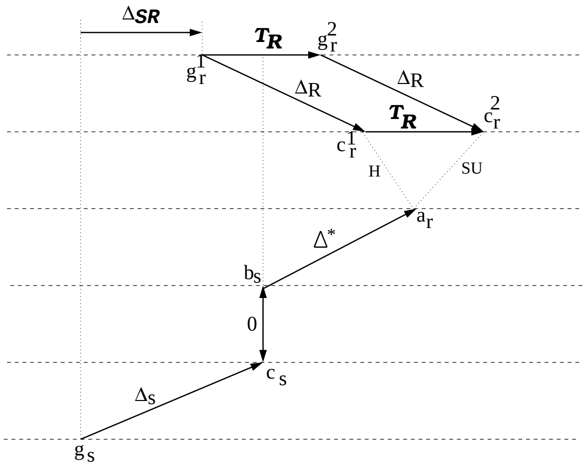
Given a GALS system, we abstract a pair of communicating LS modules as shown in Fig. 1. The simplicity of our abstraction is motivated by the set of input/output signals typically available from IP cores. Each LS module m accepts as input a clock signal, $\mathsf{G_m}$, generated by a clock generator (e.g., crystal oscillator or ring oscillator). For large IP cores, the signal $\mathsf{G_m}$ is buffered using a clock distribution tree within the core before being fed to the latches and flip-flops. We denote the buffered clock signal at the leaves of the distribution tree of module m by $\mathsf{C_m}$. Each module m is also assumed to be equipped with an input port $\mathsf{A_m}$ and an output port $\mathsf{B_m}$ that *may* be used for handshaking in a communication protocol. Output events on $\mathsf{B_m}$ are assumed to be synchronized with the buffered clock $\mathsf{C_m}$. Events on input port $\mathsf{A_m}$ are sampled by latches and/or flip-flops in module m. Therefore, these events must satisfy *setup* and *hold* time constraints with respect to the buffered clock $\mathsf{C_m}$. In Fig. 1, we assume that data is transferred from module S to module R through a data bus. The shaded block in the figure represents an abstract interface between the communicating modules. This represents the most general view of an interface between LS modules in a GALS system.

In general, events on all the above signals, namely $\mathsf{G_R}$, $\mathsf{G_S}$, $\mathsf{C_R}$, $\mathsf{C_S}$, $\mathsf{A_R}$, $\mathsf{A_S}$, $\mathsf{B_R}$ and $\mathsf{B_S}$, are observable by the interface, while only $\mathsf{A_R}$ and $\mathsf{A_S}$ are controllable by the interface. In specialized interfacing schemes like pausible clocking [1], signals $\mathsf{G_S}$ and $\mathsf{G_R}$ are also controllable by the interface. Note that our assumption about the presence of input and output handshaking ports in each LS module does not reduce the generality of our abstract model. Depending on the communication protocol, the ports may or may not be used by a pair of communicating modules.

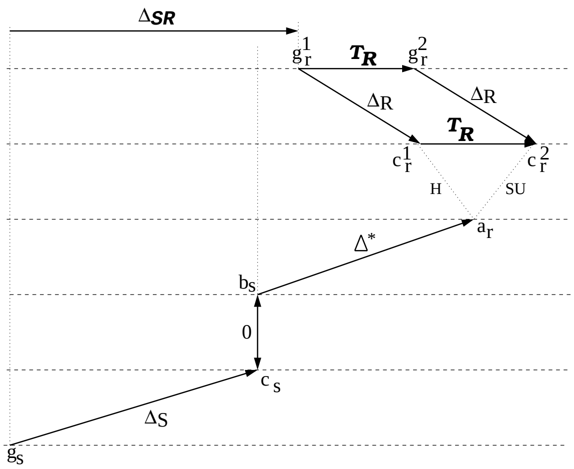
Given the above model of an interface, we use a graphical notation to represent snapshots of events of interest and timing relations between them. Our notation may be viewed as an abstraction of waveform diagrams with causality and timing relations explicitly indicated on the diagrams. We represent evolution of time along the horizontal axis; thus, time flows from left to right. A separate horizontal line is allocated for each control, data or clock signal that participates in the interfacing protocol (see Fig. 1). For the current discussion, however, we assume that changes on data lines occur synchronously with changes on handshaking signals at the interface of LS modules. Thus, it suffices to restrict our attention to the timing of control signals. An event, $\mathsf{e_x}$, of interest on signal x is represented by a point labeled $\mathsf{e_x}$ on the horizontal line corresponding to x. If an event $\mathsf{e_x}$ triggers an event $\mathsf{e_y}$ after a finite delay Δ, we draw a solid arrow labeled Δ from the point representing $\mathsf{e_x}$ to that representing $\mathsf{e_y}$. The point representing $\mathsf{e_y}$ is also plotted to the right of that representing $\mathsf{e_x}$ to denote passage of time. Note that the numerical value of Δ may not be known a priori; in such cases, we can treat Δ as a symbolic delay. If two events (on different signals) are known to occur close enough in time to be treated as simultaneous, the corresponding points are plotted vertically aligned, and a solid bidirectional arrow labeled 0 is drawn



2(a)



2(b)



2(c)

Fig. 2. Abstract timing diagrams of the generic interface

between them. The evolution of an unspecified amount of time between two events on the same signal is denoted by a dotted line between the corresponding points.

For clarity, we group the horizontal lines for signals to and from the receiver module by drawing them adjacent to each other. A similar grouping is done for signals to and from the sender module. A convenient way of grouping these lines is shown in Fig. 2. Here, the horizontal lines for clock generators $\mathsf{G_R}$ and $\mathsf{G_S}$ are spaced maximally apart since they do not interact with each other directly. The lines corresponding to $\mathsf{A_R}$ and $\mathsf{B_S}$, though belonging to different modules, are drawn close to each other because an event on $\mathsf{B_S}$ (signifying a request from the sender) may trigger an event on $\mathsf{A_R}$ (signifying a request to the receiver), depending on the handshaking protocol. It is important to note that we do not represent all events on a signal along the corresponding horizontal line. Instead, only events of interest are represented, while others are abstracted out.

An abstract timing diagram obtained using the above rules is simply a graphical representation of a partial order between the occurrence times of events of interest on the interface signals shown in Fig. 1. Given an interfacing scenario, we first select a set of events of interest on the interface signals. The occurrence times of events $\mathsf{e_x}$ and $\mathsf{e_y}$ are then said to be partially ordered if $t(\mathsf{e_x}) \leq t(\mathsf{e_y})$, where $t(\mathsf{e})$ denotes the occurrence time of event e. For every such partially ordered pair of occurrence times, a solid edge is drawn from the point representing $\mathsf{e_x}$ to that representing $\mathsf{e_y}$. The label of the edge is assigned the numeric or symbolic value of the difference $t(\mathsf{e_y}) - t(\mathsf{e_x})$. Conversely, given an abstract timing diagram, for every edge labeled Δ from $\mathsf{e_x}$ to $\mathsf{e_y}$, we can infer that $t(\mathsf{e_y}) \ = \ t(\mathsf{e_x}) + \mathsf{Delta}$.

The diagrams in Figs. 2a, b and c represent the most general scenarios of control signal timings when a message is sent from module S to module R. Let $\mathsf{b_s}$ denote the message event on output port $\mathsf{B_S}$ of module S. Since this event is generated synchronously with the buffered clock $\mathsf{C_S}$ of the sender, the only event of interest on $\mathsf{C_S}$ is the active clock edge $\mathsf{c_s}$ that triggers $\mathsf{b_s}$. Similarly, the only event of interest on the sender's clock generator, $\mathsf{G_S}$, is the active clock edge $\mathsf{g_s}$ that eventually gives rise to $\mathsf{c_s}$. The labels Δ_S and Δ_R in Fig. 2 represent the delays of clock buffering trees in the sender and receiver modules respectively. The label Δ^* on the arrow from $\mathsf{b_s}$ to $\mathsf{a_r}$ represents the delay introduced by the interface circuit after receiving message event $\mathsf{b_s}$ from the sender, and before forwarding it as event $\mathsf{a_r}$ on input port $\mathsf{A_R}$ of the receiver.

On the receiver's side, the events of interest include the forwarded message event $\mathsf{a_r}$ on input port $\mathsf{A_R}$. In addition, two active edges of the buffered clock $\mathsf{C_R}$ are of interest: the edge that samples $\mathsf{a_r}$ and the one preceding that. We represent these two active edges by points labeled $\mathsf{c_r^2}$ and $\mathsf{c_r^1}$ respectively, as shown in Figs. 2a, b and c. The events of interest on the receiver's clock generator, $\mathsf{G_R}$, are the active edges that eventually give rise to $\mathsf{c_r^1}$ and $\mathsf{c_r^2}$. These are represented by points labeled $\mathsf{g_r^1}$ and $\mathsf{g_r^2}$ respectively. The clock period between $\mathsf{g_r^1}$ and $\mathsf{g_r^2}$ is denoted $\mathsf{T_R}$. The label Δ_{SR} in Fig. 2 denotes the (possibly negative) delay from event $\mathsf{g_s}$ to event $\mathsf{g_r^1}$. The dotted lines labeled H and SU between events $\mathsf{c_r^1}$, $\mathsf{a_r}$ and $\mathsf{c_r^2}$ denote hold-time and setup-time constraints for correct sampling of $\mathsf{a_r}$ by the buffered clock of the receiver.

The design requirements for a correct interface between modules S and R can now be stated as follows.

R1: *Every message on output port* $\mathsf{B_S}$ *must be forwarded to input port* $\mathsf{A_R}$ *such that both setup and hold-time constraints are satisfied.*

R2: *If clock generator signals are also controlled by the interface circuit (like* $\mathsf{G_R}$ *in the case of pausible clocking), the rise and fall time constraints and pulse-width constraints of these signals must be met.*

Since the abstract interface in Fig. 1 can control only signals $\mathsf{A_R}$ and $\mathsf{A_S}$ in general, and additionally signals $\mathsf{G_R}$ and $\mathsf{G_S}$ in pausible clocking schemes, it follows that the above requirements are both necessary and sufficient to ensure error-free communication of a message from module S to module R. For interfacing schemes that do not rely on controlling clock generators, only requirement R1 must be met. In the following discussion, we first focus on this requirement. Subsequently, we will analyze a pausible clocking scheme in our framework and show how requirement R2 is met by careful circuit design.

Fig. 2a depicts the scenario where the forwarded message event ($\mathsf{a_r}$) is sampled by a clock edge ($\mathsf{c_r^2}$) that was generated (as $\mathsf{g_r^2}$ on $\mathsf{G_R}$) even before the original message event $\mathsf{b_s}$ occurred. In Fig. 2b, the clock edge $\mathsf{c_r^2}$ that samples $\mathsf{a_r}$ is generated (on $\mathsf{G_r}$) within one clock period after the occurrence of $\mathsf{b_s}$. Finally, Fig. 2c depicts the scenario where the clock edge that samples $\mathsf{a_r}$ is generated at least one clock period ($\mathsf{T_R}$) after $\mathsf{b_s}$ occurs. While each of these diagrams depicts a valid interfacing scenario, not all of them may be relevant for a specific interfacing scheme. For example, in a pausible clocking scheme,the scenario depicted in Fig. 2a is not very meaningful. In such a scheme, the clock is "stretched" whenever needed, to enable the forwarded message $\mathsf{a_r}$ to meet setup and hold time constraints with respect to $\mathsf{c_r^1}$ and $\mathsf{c_r^2}$. However, since the stretching action can be initiated only after event $\mathsf{b_s}$ occurs, the clock edges $\mathsf{c_r^1}$ and $\mathsf{c_r^2}$ in Fig. 2a are not affected by the stretching action. Thus, the purpose of stretching the clock is defeated; Fig. 2a is therefore not meaningful in the context of pausible clocking. It is easy to see from each of these diagrams that the following inequalities must be satisfied for error-free synchronization:

$$\Delta_{\mathsf{S}} + \Delta^* > \Delta_{\mathsf{SR}} + \Delta_{\mathsf{R}} + \mathsf{H}, \text{ and} \tag{1}$$

$$\Delta_{\mathsf{S}} + \Delta^* + \mathsf{SU} < \Delta_{\mathsf{SR}} + \mathsf{T}_{\mathsf{R}} + \Delta_{\mathsf{R}} \tag{2}$$

On rearranging, this gives

$$\Delta^* > \Delta_{\mathsf{SR}} + (\Delta_{\mathsf{R}} - \Delta_{\mathsf{S}}) + \mathsf{H}, \text{ and} \tag{3}$$

$$\Delta^* < \Delta_{\mathsf{SR}} + (\Delta_{\mathsf{R}} - \Delta_{\mathsf{S}}) + (\mathsf{TR} - \mathsf{SU}) \tag{4}$$

Inequalities (3) and (4) quantify requirement R1 stated above for a correct interface design. In order to study this requirement in some more detail, we group the delays in the above inequalities into three categories.

1. *Known but not controllable delays*: This includes Δ_{R}, Δ_{S}, SU and H, whose values are known but which cannot be controlled by the interace designer. If we aren't using a pausible clocking scheme, T_{R} also belongs to this category. If the sender and receiver clocks are related, Δ_{SR} may also be included in this category.
2. *Unknown delays*: If the communication pattern is irregular and if there is no pre-determined phase or frequency relation between the sender and receiver clocks, Δ_{SR} belongs to this category.
3. *Controllable delays*: Delays like Δ^* whose values are controllable by the interface designer fall in this category. With a pausible clocking scheme, T_{R} also belongs to this category.

Given a specific synchronization technique, we first categorize the different delays as described above, and then reason about requirements R1 and R2 for correct operation at the level of abstraction represented in the abstract timing diagrams. For example, if a synchronization technique does not control the receiver clock generator (non pausible clocking), T_{R} is a known constant that is not controllable by the interface. Accordingly, a sufficient condition for the correct operation of the interfacing scheme is obtained by treating T_{R} as a constant in inequalities (3) and (4), and ensuring that Δ^* tracks Δ_{SR}. Similarly, if there is a known relation between the sender and receiver clocks, Δ_{SR} may be treated as a known constant, and Δ^* appropriately designed to satisfy the above inequalities.

3 Abstract timing diagram based analysis of interfacing schemes

In this section, we briefly look at a few interfacing schemes in the framework of abstract timing diagrams. The analysis of other interfacing schemes can be done in a similar manner.

3.1 Pausible clocking

We first look at pausible clocking, as described by Yun et al [1], for interfacing independent clock domains. We assume, as required by Yun et al, that the clock buffering delay Δ_R is small compared to the receiver module's nominal clock period, i.e. $\Delta_R \leq T^n_R$. In this interfacing scheme,

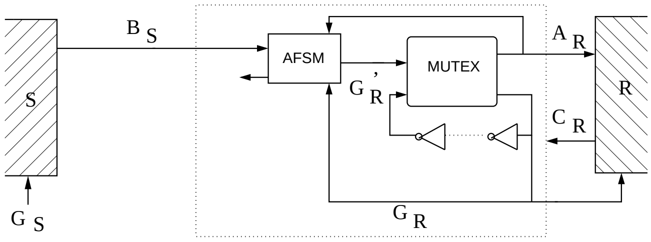
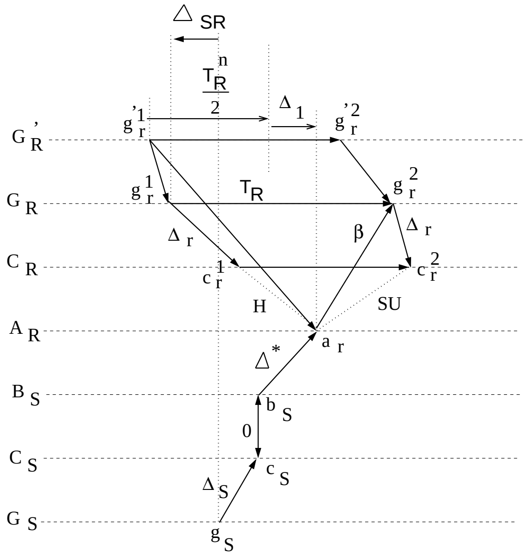


Fig. 3. Pausible clocking scheme

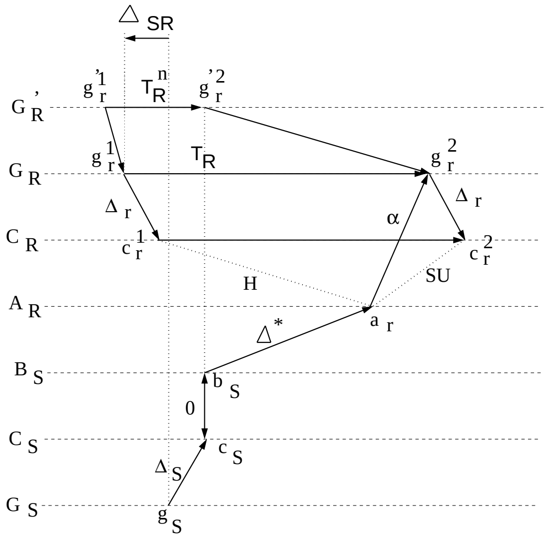
the receiver clock generator, G_R, is controlled by the interface circuitry. Specifically, the interface uses a ring oscillator, from which a signal, say G'_R is tapped. Active edges on this signal are used along with the sender's message event b_s to determine if sampling b_s by the active edge of G'_R would lead to metastability. If there is no possibility of metastability, an active edge of G_R is generated from the active edge of G'_R, and the sender's message b_s is delayed by an amount Δ^* such that both setup and hold time constraints are met. On the other hand, if there is a possibility of metastability, either the active edge of G_R is delayed or the sender's message event b_s is delayed, such that both setup and hold times are once again satisfied with respect to the active edges of G_R. The situation in terms of our abstract interface diagram is as shown in Fig. 3. Here, G_R is the generated clock signal that is fed to the receiver module, while G'_R is the signal tapped from the ring oscillator internally within the interface.

Fig. 4a shows the abstract timing diagram when there is no possibility of metastability due to sampling of b_s by $g_r^{1'}$. Depending on whether b_s



$$\beta = \Delta_3 + \Delta_2$$

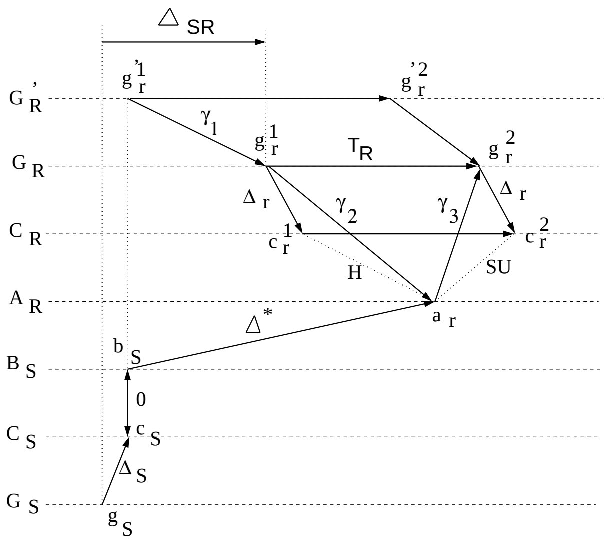
4(a)



$$\Delta^* = \Delta_m + \Delta_1$$

$$\alpha = \Delta_3 + \Delta_2$$

4(b)



$$\gamma_1 = \Delta_m + \Delta_3$$

$$\gamma_2 = \frac{T_R^n}{2} + \Delta_2 + \Delta_1$$

$$\gamma_3 = \Delta_2 + \Delta_3$$

4(c)

Fig. 4. Abstract timing diagrams for pausible clocking scheme

arrives in the active or inactive phase of G'_R, the delay Δ^* is adjusted so that the forwarded event a_r satisfies the hold-time constraint with respect to the active edge, g_r^1, of G_R. This is achieved by feeding g_r^1 and a delayed version of b_s to the inputs of a mutex element as shown in Fig. 3. If b_s arrives in the active phase of G'_R, event a_r is generated after G'_R goes low and a delay equal to the forward path delay of the mutex element has elapsed. On the other hand, if b_s arrives in the inactive phase of G_R, event a_r is generated after the forward path delay of the mutex element. Let the forward path delay of the mutex element for input B_S be Δ_1. Thus, we have $\Delta_1 \leq \Delta^* \leq \Delta_1 + \frac{T_r^n}{2}$, where T_R^n is the nominal clock period of the receiver. It is also easy to see from Fig. 4a that $\Delta_{SR} \leq \Delta_S - \Delta_R$. Inequality (3) can therefore be satisfied if we ensure that $\Delta_1 > H$.

Once a_r is asserted, an asynchronous FSM coupled with the mutex element (see Fig. 3) ensures that the next active edge, g_r^2, on G_R does not arrive (even if the next active edge $g_r^{2'}$ of G'_R has arrived) until after a certain delay has elapsed. This delay includes the response time of the asynchronous FSM, say Δ_2, and the forward delay of the mutex element for input G'_R, say Δ_3. Thus, the delay from a_r to g_r^2 in Fig. 4a is at least as large as $\Delta_2 + \Delta_3$. By choosing Δ_2 and Δ_3 sufficiently large, it is possible to satisfy the setup-time constraint. In particular, it is easy to see from Fig. 4a that if $\Delta_2 + \Delta_3 + \Delta_R$ is chosen at least as large as SU, inequality (4) is satisifed. Thus, by knowing the details of the interfacing scheme, we can derive necessary constraints on delays of circuit elements for error-free synchronization.

Figs. 4b and c depict the scenarios when the active edge of G'_R and the message event b_s occur close enough in time to lead to metastability. A pausible clocking scheme uses a mutex element to ensure that in such a scenario, there are one of two outcomes:

1. The message event b_s "wins" over the active edge of G'_R. Event b_s is therefore forwarded to the receiver as event a_r after a delay equal to the metastability resolution time, say Δ_m, plus the forward delay, Δ_1, of the mutex element. This is depicted in Fig. 4b. Note that Δ_{SR} in this case is constrained to be $\Delta_S - T_R^n$, where T_R^n is the nominal clock period of the receiver. The hold-time constraint (inequality (3)) is met by ensuring that Δ_1 is sufficiently large. Thus, even if metastability resolves after a very short time (resulting in Δ^* being equal to Δ_1, in the worst case), a sufficiently large value of $T_R^n + \Delta_1$ ensures that inequality (3) is still satisfied. After event a_r has been generated, an asynchronous FSM coupled with the mutex element ensures that the active edge on G'_R is forwarded as event g_r^2 on G_R after a delay equal

to $\Delta_2 + \Delta_3$. By ensuring that $\Delta_2 + \Delta_3 + \Delta_R$ is at least as large as the setup-time constraint, we can effectively increases T_R to satisfy inequality (4).

2. The active edge $g_r^{1'}$ "wins" over the message event b_s. This results in the generation of active edge g_r^1 on G_R after the metastability resolution time (Δ_m) plus the forward delay, Δ_3, of the mutex element. Fig. 4c depicts this scenario. After the active edge g_r^1 is generated, input G'_R of the mutex element is deasserted after a delay of $\frac{T_R^n}{2}$. The mutex then forwards event b_s to A_R after a delay of Δ_1. Thus, by choosing Δ_1 sufficiently large, the hold time constraint in inequality (3) can be satisfied. Once event b_s is forwarded as event a_r to the receiver, the same mutex and asynchronous FSM now ensure that the next active edge, g_r^2, on G_R occurs only after a delay of $\Delta_2 + \Delta_3$. Once again, by ensuring that $\Delta_2 + \Delta_3 + \Delta_R$ is at least as large as the setup time constraint, inequality (4) can be met.

Note that in the above analysis, all our timing constraints were one sided. We required $\Delta_1, \Delta_2, \Delta_3$ to be sufficiently large so that inequalities (3) and (4) could be satisfied. This shows that by designing a slow enough interface based on pausible clocking, it is possible to guarantee error-free communication between independent clock domains.

Since a pausible clocking scheme also modifies the clock signal fed to the receiver module, we must also ensure that requirement R2 mentioned above is satisfied. In the case of Yun et al's design, this is ensured because the rise and fall of G_R are simply transitions on an output of a mutex element. Thus, by using mutex element designs that always give "clean" high or low levels at their outputs with acceptable rise and fall times, we can be assured of proper rising and falling transitions on G_R. The minimum pulse width requirement is satisfied by making sure that whenever the pulse width is modified, it is increased beyond the nominal value. Thus, using abstract timing diagrams and some details of the pausible clocking scheme, it is possible to reason about its correctness.

3.2 Pausible clocking with large clock buffer delays

An interesting situation arises when Δ_R is larger than the nominal clock period of the receiver module. If we further assume that the rate of sending of message events b_s is not known a priori, then pausing of the receiver clock can be initiated only after an event on B_S has actually occurred. As explained above, the primary purpose of pausing the clock is to ensure that the forwarded request a_r meets the setup and hold time constraints with respect to the active edges on $\mathsf{C_R}$. However, referring to Fig. 4, we note that even if the receiver clock generator is stopped τ units of time after $\mathsf{b_s}$ occurs, the buffered clock of the receiver doesn't stop until time $\tau + \Delta_R$ units of time have elapsed. Thus, in order to guarantee that setup and hold time constraints are met by a pausible clocking scheme, we need to delay the forwarded request by an amount of the order of Δ_R. This points to an inherent limitation of the pausible clocking scheme when applied to high-speed systems with large clock buffer trees in IP cores.

If we assume a full handshake protocol between the sender and receiver modules, every message sent from the sender must be acknowledged by the receiver before the sender sends the next message. Thus, inserting a delay of the order of Δ_R in the forward path (request from sender to receiver) and a delay of the order of Δ_S in the backward path (to ensure correct sampling of the acknowledge by the sender) results in the maximum overall message throughput being limited to $\frac{1}{\Delta_\mathsf{S}+\Delta_\mathsf{R}}$! Unfortunately, as our analysis indicates, this is the price that must be paid to guarantee error-free communication. Of course, if the rate of sending messages is low enough to meet this requirement, then the above scheme can be used to ensure error-free communication between two LS modules without affecting the performance.

An alternative to paying the price in terms of reduced throughput is to use an asynchronous queue between the sender and the receiver. Such an asynchronous queue could be used to buffer a series of messages sent from the sender and also to provide acknowledges to the sender, even when the receiver has not received or acknowledged a forwarded message. However, if the average rate of generation of messages is higher than the average rate at which they can be consumed by the receiver, the queue will occasionally become full. In such a scenario, it is necessary hold back acknowledges to the sender, so that no further messages are generated. Similarly, when a space is created in the queue, new acknowledges must be generated for the sender. The detection of full and non-full conditions in an asynchronous queue depend on both the insertion and deletion operations. Since these operations are synchronized with respect to the sender and receiver clocks respectively, the generation of these conditions is asynchronous with respect to the sender's clock, in general. Thus, the correct sampling of these conditions by the sender module results in timing constraints similar to those in inequalities (3) and (4). The addition of a queue therefore does not mitigate the problems inherent in synchronization.

3.3 Mesochronous, plesiochronous and multi-synchronous systems

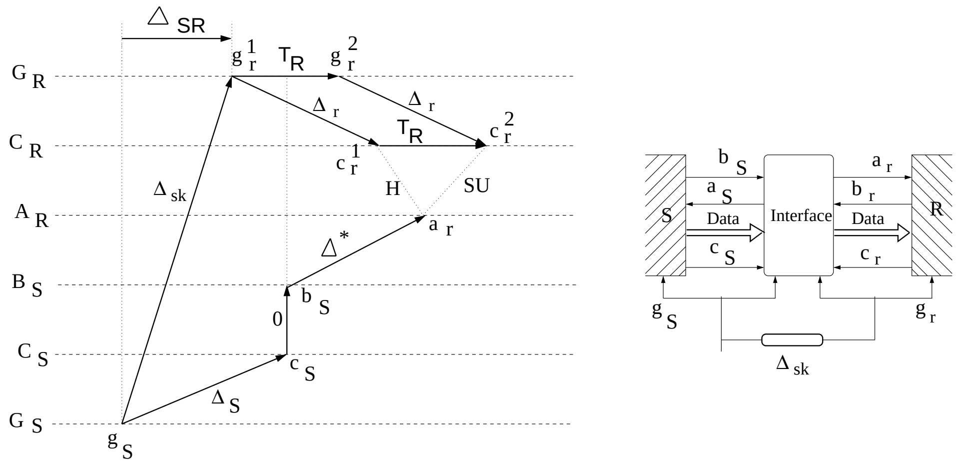


Fig. 5. Abstract timing diagram for mesochronous systems

Existing synchronization techniques for mesochronous, plesiochronous and multi-synchronous systems do not control the clock generators of the sender or receiver modules. Thus, we only need to ensure that requirement R1 for correct communication between modules is satisfied.

In mesochronous systems, a fixed relation between the sender and receiver clocks is known a priori. This is usually in the form of a known skew between the two clocks. The situation is as depicted in Fig. 5, where Δ_{sk} denotes the skew between the sender and receiver clocks. In this case, it is always possible to choose a value of Δ^* that satisfies both inequalities (3) and (4). For example, consider the situation where we choose Δ^* to lie in the range $(\alpha + \mathsf{H},\ \alpha + \mathsf{T_R} - \mathsf{SU})$, where α equals $(\Delta_{\mathsf{sk}} + n.\mathsf{T_R} + \Delta_{\mathsf{R}} - \Delta_{\mathsf{S}})$ and n is any integer ≥ 0. It can then be verified that both inequalities (3) and (4) hold with Δ_{SR} equal to $\Delta_{\mathsf{sk}} + n.\mathsf{T_R}$. This, for example, is the idea behind the correctness of STARI [7].

In plesiochronous systems, the frequencies of the sender and receiver modules are close enough, though not exactly matched. The delay Δ_{SR} is therefore not constant and drifts with time. Nevertheless, if we assume that a forwarded event $\mathsf{a_r}$ is sampled using a receiver clock edge that was generated either immediately after $\mathsf{b_s}$ occurred or one clock period after that (see, for example, Figs. 2 b and c), it is easy to see that Δ_{SR} varies in a window of 2.$\mathsf{T_R}$ from $\Delta_{\mathsf{S}} - \mathsf{T_R}$ to $\Delta_{\mathsf{S}} + \mathsf{T_R}$. If Δ_{SR} is negative but outside the setup-hold window around 0, we need to choose a small Δ^* so that inequalities (3) and (4) are satisfied. If Δ_{SR} is positive but outside the setup-hold window around 0, a larger value of Δ^* needs to be chosen to satisfy the setup and hold inequalities. Existing techniques for interfacing plesiochronous systems [13] essentially perform this adjustment of Δ^* by using a latch pulse generator circuit along with a miss-detector to generate appropriate latching pulses [13]. If, however, Δ_{SR} is within the setup and hold window around 0, a miss-detector may not function correctly, and this could lead to Δ^* not being assigned the right value leading to a sampling error. This is precisely the situation under which existing interfacing techniques for plesiochronous systems [13] malfunction.

In multi-synchronous systems, the frequencies of the sender and receiver modules are matched, but the skew Δ_{sk} between them could be varying slowly with time. Clearly, an interfacing solution for such systems must be similar to that for mesochronous systems, with the additional feature of Δ^* tracking the time-varying skew Δ_{sk}. This is precisely what is done in adaptive synchronization techniques [8]. However, if the tracking is not done correctly, and either inequality (3) or (4) is violated, a synchronization error cannot be avoided.

4 Exploring new interfaces using abstract timing diagrams

Having discussed existing interfacing schemes in the framework of abstract timing diagrams, we now proceed to see how such diagrams can help us in designing new interface circuits. We have already seen above that our analysis indicates the difficulty of scaling up existing pausible clocking techniques when the clock buffering delay is large compared to the nominal clock period. In this section, we identify a few circuit features that can enable us to interface independent clock domains in a robust way.

We refer back to Fig. 2b for our current discussion. We notice from this figure that if $\mathsf{b_s}$ occurs close enough to $\mathsf{c_s^1}$, a potential metastabile situation could result if we forward $\mathsf{b_s}$ to $\mathsf{a_r}$ too soon. If, however, we have a mechanism that guarantees that whenever such a condition arises, we generate $\mathsf{a_r}$ late enough to satisfy the hold-time constraint with respect to $\mathsf{c_r^1}$ and early enough to satisfy the setup-time constraint with respect to $\mathsf{c_r^2}$, then we can bypass the metastability-induced problem. A mutual exclusion (mutex) element is the natural choice for implementing such a mechanism. Thus, we can feed $\mathsf{B_S}$ and $\mathsf{C_R}$ as the two contending inputs to our mutex element, and we use the grant output corresponding to the $\mathsf{B_S}$ input as $\mathsf{A_R}$ to be fed to the receiver. If we do not pause the receiver clock, we are guaranteed that $\mathsf{C_R}$ deasserts itself $\frac{\mathsf{T_R^n}}{2}$ time units after the generation of $\mathsf{c_r^1}$. If a request on $\mathsf{b_s}$ was already pending, it is now guaranteed to be granted on $\mathsf{A_R}$ within a delay (Δ_1) equal to that of the forward delay of the mutex element. The requirement about setup time being satisfied with respect to the next active edge $\mathsf{c_r^2}$ on $\mathsf{C_R}$ can thus be met by designing Δ_1 to be sufficiently small. However, a problem arises if both $\mathsf{c_r^1}$ and $\mathsf{b_s}$ arrive at nearly the same time leading to a metastable condition in the mutex element. In this case, if metastability resolves too soon *and* in favour of $\mathsf{b_s}$, an event violating the hold-time constraint with respect to $\mathsf{c_r^1}$ can be generated on $\mathsf{A_R}$. In order to prevent this, we need a mutex element satisfying one of the following properties:

- Whenever a metastable condition is reached, the mutex always grants the request in favour of a chosen input. In our case, we could ensure that the request is always granted in favour of $\mathsf{C_S}$, so that an early event on $\mathsf{A_R}$ cannot be generated.
- Alternatively, if the mutex element is such that the minimum delay required for metastability to resolve is greater than $\frac{\mathsf{T_R}}{2}$, we can ensure that an early event on $\mathsf{A_R}$ is never generated. Note that this requirement is different from that of an arbiter having a bounded arbitration time. Note also that once $\mathsf{C_R}$ is deasserted (after a delay of $\frac{\mathsf{T_R}}{2}$), a pending request on $\mathsf{B_S}$ is always granted on $\mathsf{A_R}$.

While we have not been able to design a mutex element with these properties, our analysis points to directions in which circuit design research could be focussed if we are to come up with a robust solution to the interfacing problem.

4.1 An interfacing solution for high-speed SoCs

Taking cue from our abstract timing diagram based reasoning, we have designed and implemented a circuit for interfacing independent clock domains in high-speed SoCs. For simplicity, we have assumed unidirectional data transfer between two large IP cores running at frequencies between 1 and 3 gigahertz. Since large cores fed by high-speed clocks require non-trivial clock buffering within the cores, we have assumed that Δ_S and Δ_R are large compared to the sender and receiver clock periods respectively. We have also assumed a sender initiated communication protocol

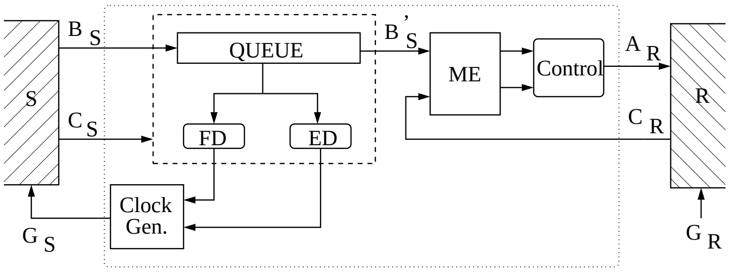


Fig. 6. Our interfacing scheme

using a partial handshake. Thus, when the sender wishes to communicate, it asserts signal $\mathsf{B_S}$ and assumes that the receiver correctly receives this request on $\mathsf{A_R}$. The sender does not wait for the receiver to acknowledge before sending its next request. The receiver, on the other hand, waits for a request on $\mathsf{A_R}$. On sampling such a request, it processes the request without sending an acknowledgement back to the sender. Thus, there is a partial handshake between the sender and receiver modules. If the buffered clocks at the sender and receiver ends are perfectly synchronized, such a protocol can allow very high rates of communication. On the other hand, if the communicating modules are not perfectly synchronized, there is a possibility of erroneous communication and loss of messages. Our motivation for choosing this protocol was to study the implications of a high-performance protocol on the design of interface circuitry. Fig. 6 shows the block-level diagram of our interfacing scheme.

We have used an asynchronous queue with a full detector (FD) and an empty detector (ED), as outlined in Section 3.2. This allows the sender to continue sending messages until the queue becomes full, even when the receiver is slow to respond. As explained in Section 3.2, when a full queue is detected, the clock generator, $\mathsf{G_R}$, of the sender must be paused to prevent queue overflow. However, since the buffered clock, $\mathsf{C_R}$, stops Δ_S time units after $\mathsf{G_R}$ is paused, adequate space must be reserved in the queue to buffer messages sent during the *clock overrun*, i.e., after $\mathsf{G_R}$ is paused but before $\mathsf{C_R}$ actually pauses. This is achieved by maintaining two queues, $\mathsf{Q_1}$ and $\mathsf{Q_2}$, such that $\mathsf{Q_1}$ is nominally used to buffer messages from the sender. However, when $\mathsf{Q_1}$ becomes full and the sender wants to send a new message, the clock generator $\mathsf{G_R}$ is paused and messages sent during the clock overrun are steered into queue $\mathsf{Q_2}$. Of course, the order of messages is preserved, so messages in $\mathsf{Q_1}$ are forwarded to the receiver before those in $\mathsf{Q_2}$. Once $\mathsf{Q_2}$ becomes empty, and is detected by ED, we release the clock generator G_R again. Thus, the clock generator of the sender is controlled by the outputs of both FD and ED. Unfortunately, since these signals are asynchronous with respect to G_R, there is a non-zero probability of bad pulses being generated on G_R. This constitutes one of the failure scenarios of our design.

On the receiver's side, we have used the idea described in Section 4 section for synchronizing A_R with C_R. In particular, we do not pause the receiver's clock. Instead, a delayed version of B_S, called B'_S, is fed to a mutex element (ME), whose other input is fed by the buffered clock C_R (see Fig. 6). As explained in Section 4, if an event on B'_S arrives outside the setup and hold window of C_R, the forwarded event a_r can be guaranteed to meet the setup and hold time requirements with respect to C_R. However, if B'_S changes too close to C_R, the mutex element might get into metastability, resulting in a non-zero probability of failure. However, if we could design a mutex element with the properties explained earlier, this scheme would guarantee error-free synchronization.

We have designed the interface circuit described here and performed SPICE simulations at all process corners and temperatures ranging from 0^oC to 70^oC. While our simulations do not indicate any failures, a detailed transistor-level timing analysis shows the same problems as uncovered by our abstract timing diagram based reasoning. These results affirm our belief that abstract timing diagrams are useful tools for reasoning about synchronization issues in GALS interfaces at a high-level of abstraction.

5 Conclusion

Interface design will continue to be a major challenge as we move into the era of large SoCs with multiple high-speed IP cores fed by independent clocks. Designing interfaces that ensure correct communication between modules without adversely affecting the overall system performance are crucial for the development of future SoCs. While several interfacing solutions have been proposed in the literature, there has been a lack of uniform analysis techniques that allow one to compare the strengths and weaknesses of different techniques, and help designers in improving existing solutions and coming up with new ones. In this paper, we presented abstract timing diagrams to reason about synchronization issues in several interface designs at a high level of abstraction. The advantage of these diagrams over detailed timing diagrams is that they allow us to focus on crucial timing requirements between interface signals without getting bogged down with implementation details. If details of a specific implementation are known, it is possible to refine these diagrams further to verify whether high-level timing constraints are actually met by the lower level implementation. For example, we showed that it is possible to reason about the correctness of pausible clocking when clock buffering delays are small by using some implementation details of a pausible clocking scheme. We also briefly discussed how it is possible to reason about the correctness of other interfacing schemes like STARI and adaptive synchronization in this framework. We demonstrated how our framework allows us to compare different interfacing solutions and discover their limitations. Finally, we showed that our analysis shows that if mutex elements with certain properties can be designed, we can solve the interfacing problem elegantly. Whether such circuits can be designed remains, of course, an open question. We believe that abstract timing diagrams will serve as a useful aid to interface designers when reasoning about the correctness of their designs even prior to implementation.

References

1. K. Y. Yun and R. P. Donohue, "Pausible clocking: A first step toward heterogeneous systems," in *Proc. International Conf. Computer Design (ICCD)*, 1996.
2. D. M. Chapiro, *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, October 1984.
3. J. Muttersbach, *Globally-Asynchronous Locally-Synchronous Architechtures for VLSI Systems*. PhD thesis, ETH Zurich, 2001.
4. A. E. Sjogren and C. J. Myers, "Interfacing synchronous and asynchronous modules within a high-speed pipeline," in *Advanced Research in VLSI*, pp. 47–61, 1997.
5. P. W. Joep Kessels, Ad Peeters and S.-J. Kim, "Clock synchronization through handshake signalling," in *Proc. International Symposium on Asynchronous Circuits and Systems*, April 2002.
6. N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI design, a systems perspective*. Addison-Wesley Publishing Company, second edition 1992.
7. M. R. Greenstreet, *STARI: A Technique for High-Bandwidth Communication (Thesis)*. PhD thesis, 1993.
8. R. Ginosar and R. Kol, "Adaptive synchronization," in *Proc. Internanational Conf. Computer Design (ICCD)*, pp. 188–189, October 1998.
9. K. Y. Yun and A. E. Dooply, "Pausible clocking based heterogeneous systems," in *IEEE Transactions on VLSI systems*, pp. Vol. 7, no. 4 : 482 – 487, December 1996.
10. S. M. Burns, *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1991.
11. S. Chakraborty, *Polynomial-Time Techniques for Approximate Timing Analysis of Asynchronous Systems*. PhD thesis, Stanford University, Aug. 1998.
12. C. J. Myers and T. H.-Y.Meng, "Synthesis of timed asynchronous circuits," in *IEEE Transactions on VLSI Systems*, pp. 1(2):106–119, June 2002.
13. A. Chakraborty and M. R. Greenstreet, "Efficient self-timed interfaces for crossing clock domains," in *Proc. of the Nineth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, May 2003.