Reasoning about Synchronization in GALS Systems

Supratik Chakraborty, Joycee Mekie and Dinesh K. Sharma
Indian Institute of Technology, Bombay, Mumbai 400076, INDIA

Abstract. Correct design of interface circuits is crucial for the development of System-on-Chips (SoC) using off-the-shelf IP cores. For correct operation, an interface circuit must meet strict synchronization timing constraints, and also respect sequencing constraints between events dictated by interfacing protocols and rational clock relations. In this paper, we propose a technique for automatically analyzing the interaction between independently specified synchronization constraints and sequencing constraints between events. We show how this analysis can be used to derive delay constraints for correct operation of interface circuits in a GALS system. Our methodology allows an SoC designer to mix and match different interfacing protocols, rational clock relations and synchronization constraints for communication between a pair of modules, and automatically explore their implications on correct interface circuit design.

1. Introduction

Recent advances in VLSI technology have made possible the integration of complex subsystems like microprocessors, digital signal processors and embedded controllers on a single die, giving rise to System-on-Chip (SoC) designs. Globally Asynchronous Locally Synchronous (GALS) architectures offer a promising approach to manage the complexity of such designs. Specifically, GALS architectures are ideally suited for SoCs designed with off-the-shelf IP cores. Individual cores like microprocessors can be viewed as *locally synchronous* modules (henceforth called *LS* modules), while communication between modules, potentially clocked at different frequencies, can be viewed as asynchronous interactions. The primary task of an SoC designer is therefore centred around integrating and interfacing different LS modules in a single chip to implement the overall functionality of the system. Tools and techniques for reasoning about the correctness and performance of interfaces are therefore crucial for successful SoC design. This motivates our current work.

While interface design is not a new discipline, several new challenges arise when interfacing IP cores in modern SoCs. For example, building protocol converters that allow IP cores with incompatible handshaking protocols to communicate with each other is an important activity in SoC design. However, this problem has been discussed extensively in earlier work (e.g. [24]), and *is not the focus of this paper*. For purposes of our discussion, we will assume that communicating modules (possibly after insertion of protocol converters) always agree on the sequence of handshaking events. Our focus will, instead, be on the interaction between synchronization and sequencing constraints when LS modules communicate with each other. Long and transiently varying interconnect delays are known to cause synchronization problems in high-speed SoCs when signals cross LS module boundaries. If individual modules are clocked at different frequencies with phase and frequency jitters, inter-module communication is vulnerable to synchronization failures even with short and predictable interconnect delays. To make matters worse, certain interfacing schemes, like stoppable clocking [25, 27, 28, 23, 8, 7, 31], that work perfectly well at low clock frequencies may malfunction at higher frequencies if communicating modules have large internal clock buffering delays [18]. Similarly, an error-free interface may become vulnerable to synchronization failures if the ratio of clock frequencies of communicating modules changes or if interfacing protocols are changed slightly. Thus, designing correct interfaces presents a spectrum of issues even when handshaking protocols agree on the sequence of events. In this paper, we address one such issue. Specifically, we describe a methodology for analyzing the interaction between independently specified synchronization and sequencing constraints. This analysis is then used to derive delay constraints for correct operation of an interface circuit. We believe our analysis is most useful prior to detailed circuit design, when an SoC designer may wish to explore the implications of mixing and matching different interfacing protocols, clock frequency ratios and synchronization constraints to implement communication between LS modules.

We propose *abstract timing diagrams* (ATDs) as a graphical formalism for specifying synchronization constraints between interface events. We also present finite-state *sequence diagrams* for specifying sequencing constraints between events arising out of interfacing protocols and rational clock relations. While timing constraints represented by ATDs can be analyzed directly to obtain delay constraints for correct operation of an interface circuit, it is often beneficial to derive less conservative constraints by taking into account the interaction of synchronization and sequencing constraints. The primary contribution of this paper is to propose a methodology for achieving this.

While representations similar to ATDs have been used earlier [1, 17, 30, 3, 6, 21, 12], applying these earlier approaches to our problem would require the designer to manually (or otherwise) identify all possible scenarios of interaction between sequencing and synchronization constraints for a given interface, and then represent them explicitly in a combined diagram. This poses practical difficulties in a design-analyze-redesign setting, where a designer may wish to go through several iterations of modifying sequencing and synchronization constraints. For example, consider a correctly functioning SoC in which we wish to change the interfacing protocol between two communicating modules, possibly for performance reasons. In order to determine if all synchronization constraints will continue to be satisfied in the modified system, the designer must first determine all possible interactions between the changed sequencing constraints (due to the changed interfacing protocol) and synchronization constraints. She must then redraw all timing diagrams to explicitly indicate these new interactions, and finally analyze the modified diagrams. We believe this approach has several drawbacks. First, it is natural for a designer to think independently about synchronization constraints, interfacing protocols and rational clock relations. Hence, from a designer's perspective, it is desirable to logically separate the specification of synchronization constraints from the specification of sequencing constraints. In addition, since the exploration of interactions between sequencing and synchronization constraints is left to the designer, the likelihood of corner-case scenarios being missed is increased. This adversely affects the effectiveness of existing approaches in uncovering potential timing problems, when incremental changes are made to sequencing and/or synchronization constraints. The present work addresses both these drawbacks. First, we give the designer the freedom to separately specify synchronization and sequencing constraints. In addition, we provide an automatic technique that analyzes all synchronization constraints in appropriately chosen sequences of events allowed by the sequencing constraints. This removes the possibility of errors creeping in due to oversight during manual analysis of the interaction between sequencing and synchronization constraints.

The literature contains an impressive collection of interfacing schemes for GALS systems [25, 27, 23, 7, 8, 31, 20, 26, 16]. While some schemes, like those based on two-flop synchronizers [29], are applicable in a general setting, they suffer from non-zero probability of failure. Other schemes like STARI [14], adaptive synchronization [13], schemes for plesiochronous systems [4] or stoppable clocking [25, 27, 28, 23, 8, 7, 31] guarantee failure-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 its own independent analysis of various synchronization scenarios and failure modes (if any). Since the analysis methodology presented in this paper allows independent specification of synchronization and sequencing constraints, we note in passing that our methodology makes it possible to reason about diverse interfacing schemes within the same framework.

The remainder of this paper is organized as follows. Section 2 introduces ATDs and shows how necessary and sufficient constraints for correct operation of an interface circuit can be derived from these diagrams. Section 3 introduces sequence diagrams as a formalism for specifying sequencing constraints between events, and presents a technique for automatically checking synchronization constraints specified by ATDs in appropriate sequences of events allowed by a sequence diagram. Section 4 compares our work with

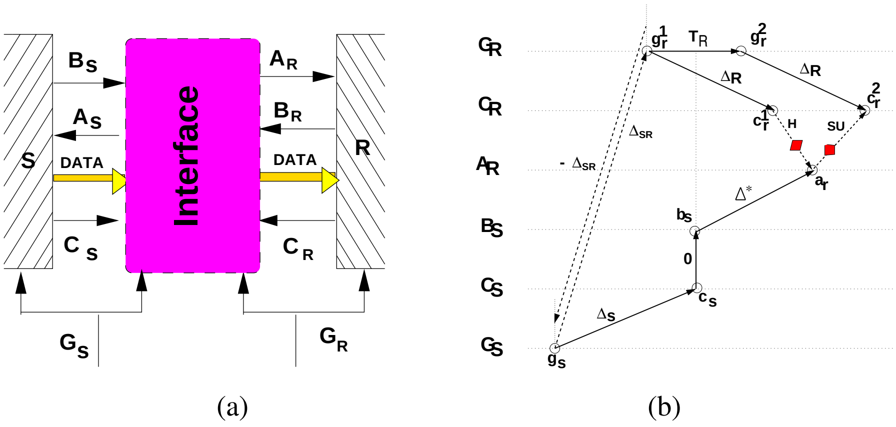

Figure 1. (a) A generic interface in a GALS system, and (b) Abstract Timing Diagram

two earlier ones that come closest to our work. Finally, section 5 concludes the paper, and points to future directions of research.

2. Abstract Timing Diagrams

An *Abstract Timing Diagram*, or ATD, is a graphical notation for representing abstract events and timing relations between them. ATDs are syntactically similar to commonly used waveform diagrams, timing diagrams [1, 12], event-rule systems [3, 21], event graphs [30], action diagrams [17], timing constraint graphs [6] and message sequence charts [15]. However, the analysis of interface timing constraints using ATDs differs from earlier approaches, as will be elaborated in Section 3.

We will use a simple example to develop the intuition behind the construction and use of ATDs. Consider a GALS system with two LS modules clocked at different frequencies and communicating with each other through an interface circuit, as depicted in Fig. 1(a). Each LS module, M, (in Fig. 1(a), M is either S or R) accepts as input a clock signal, $\mathsf{G_M}$, generated by a clock generator (e.g., crystal oscillator or ring oscillator). The signal $\mathsf{G_M}$ is then buffered using a clock distribution tree within the module before being fed to flip-flops. We denote the buffered clock signal by $\mathsf{C_M}$. Each module M also has an input port $\mathsf{A_M}$ and an output port $\mathsf{B_M}$ that are used for handshaking during communication. Data is transferred from module S to module R, and changes in data lines are assumed to be synchronized with changes in handshaking signals at the interface of the two modules. Thus, it suffices to restrict our attention to the timing of only handshaking and clock signals.

To describe an interfacing scenario of the above system using an ATD, we must first identify a set of *abstract events* that are relevant for describing the scenario. In general, an abstract event is an action on a signal or port (henceforth simply called a "signal") at an appropriately chosen level of abstraction. All abstract events are considered to be *atomic and instantaneous*. Thus, an abstract event cannot occur partially – it either occurs or it does not. Similarly, the initiation and completion of an abstract event are regarded as occurring at the same time. In the system of Fig. 1(a), if we assume that all sampling elements are positive edge-triggered flip-flops, and if we use the binary abstraction of signals, the set of abstract events includes all 0 to 1 and 1 to 0 changes of $\mathsf{A_S}$, $\mathsf{B_S}$, $\mathsf{A_R}$ and $\mathsf{B_R}$, and all 0 to 1 changes of $\mathsf{C_S}$, $\mathsf{C_R}$, $\mathsf{G_S}$ and $\mathsf{G_R}$. Note that we have ignored the 1 to 0 transitions of all clocks since all clocked elements are assumed to be positive edge-triggered.

When drawing an ATD, we represent evolution of time along the horizontal axis. A separate line is drawn for each signal that has an abstract event (henceforth simply called an "event"). An event e on signal x is represented by a square or a circular vertex labeled e on the horizontal line corresponding to x. Square vertices represent events with AND causality, while circular vertices represent events with OR causality. The ability to model events with both AND and OR causality is useful when reasoning about gate level interface circuits. Events on the same signal are plotted in chronological order from left to right on the horizontal line corresponding to the signal. If event $\mathsf{e_2}$ is plotted to the immediate right of event $\mathsf{e_1}$ on the horizontal line corresponding to signal x, then $\mathsf{e_1}$ and $\mathsf{e_2}$ represent two consecutive events on x. Causal relations and known timing relations between events are explicitly represented with labeled, directed edges between the corresponding vertices. A solid edge labeled $\Delta_{\mathsf{e,f}}$ from e to f represents a delay of $\Delta_{\mathsf{e,f}}$ between the occurrence of event e and the enabling of event f by event e. Let $t(\mathsf{e})$ denote the time of occurrence of event e, and $preds(\mathsf{e})$ denote the set of vertices from which there exist solid edges to e. If e is represented by a square vertex, then $t(\mathsf{e})$ is given by $\max_{r \in preds(\mathsf{e})} \; (t(\mathsf{r}) + \Delta_{r,e})$; otherwise, $t(\mathsf{e})$ equals $\min_{r \in preds(\mathsf{e})} \; (t(\mathsf{r}) + \Delta_{r,e})$. We also use dashed, directed edges to represent difference constraints between times of occurrence of events. Thus, a dashed edge labeled $\Delta_{\mathsf{e,f}}$ from e to f represents the constraint $t(\mathsf{f}) - t(\mathsf{e}) \;\geq\; \Delta_{\mathsf{e,f}}$.

The set of min, max and linear constraints represented by an ATD can be divided into two categories: (i) *assume* constraints, that arise from the operation of the system and may be assumed to be satisfied, and (ii) *guarantee* constraints, that must be enforced for the correct functioning of the system. For example, the difference constraint between two successive ticks of a free-running clock is an assume constraint, while *setup* and *hold-time* constraints of sampled signals are guarantee constraints. We represent guarantee constraints explicitly in an ATD by annotating the corresponding edges with solid diamonds. Thus, the overall set of assume (guarantee) constraints represented by an ATD are obtained by removing all edges with (without) diamonds, and conjoining the constraints represented by the remaining edges.

Since solid edges represent causal relations between events, their delay labels are always non-negative. Dashed edges, on the other hand, may have negative delay labels, since they represent difference (and not necessarily causal) constraints. The edge labels themselves can either have numerical delay values or be treated as symbolic delays. This makes it possible to represent symbolic timing constraints using ATDs.

Fig. 1(b) shows an ATD depicting an interfacing scenario for the system of Fig. 1(a). In this scenario, a request b_s is sent on port B_S from module S to module R. Since the request is generated synchronously with the buffered clock, C_S, the only relevant event on C_S is the rising clock transition, c_s, that triggers b_s. Similarly, the only event of relevance on the sender's clock generator, G_S, is the rising transition g_s that gives rise to c_s. On the receiver's side, the relevant events include the forwarded request a_r, and two events on the buffered clock C_R: the rising clock transition, c_r^2, that samples a_r and the rising transition, c_r^1, preceding that. The events of relevance on the receiver's clock generator, G_R, are the transitions g_r^1 and g_r^2 that give rise to c_r^1 and c_r^2, respectively. The clock period between g_r^1 and g_r^2 is denoted as T_R. The label Δ_{SR} in Fig. 1(b) denotes the (possibly negative) delay from event g_s to event g_r^1. The labels Δ_S and Δ_R represent clock buffering delays in the sender and receiver modules respectively. The label Δ^* on the edge from b_s to a_r represents the delay introduced by the interface circuit.. The dashed lines annotated with diamonds and labeled SU and H represent setup and hold-time constraints, respectively, for correct sampling of a_r. These are the only guarantee constraints in Fig. 1(b); all other constraints are of the assume type.

Note that the ability to represent symbolic timing constraints between events is a powerful feature, and distinguishes ATDs as also the work of Amon et al [1], Fisler [12] and Clariso et al [9] from several earlier work [3, 17, 30, 6, 21]. The ability to represent assume and guarantee constraints of min, max and linear types in the same diagram makes it possible to perform more detailed analysis using ATDs than is possible with the representations used in [3, 21, 6, 15]. In addition, the goal of our ATD-based analysis is to derive symbolic delay constraints for correct operation of interface circuits. This contrasts with the work of [3, 21, 6, 30, 17], where the goal is to compute performance metrics and bounds on time separation of events. The work that comes closest to our are those of Amon et al [1] and Clariso et al [9]. A detailed comparison of our approach with these two earlier work is provided in Section 4.

2.1. Deriving delay constraints from ATDs

We have seen above that edges in an ATD represent linear, max and min constraints between times of occurrence of events. A max or min constraint can, in turn, be expressed as a Boolean combination of linear inequalities. For example, the constraint $t(\mathsf{f}) = \max_{e \in preds(f)}(t(\mathsf{e}) + \Delta_{e,f})$ is equivalent to

$$\left(\bigwedge_{e \in preds(f)} (t(\mathsf{f}) \geq t(\mathsf{e}) + \Delta_{e,f})\right) \wedge \left(\bigvee_{e \in preds(f)} (t(\mathsf{f}) \leq t(\mathsf{e}) + \Delta_{e,f})\right).$$

A min constraint can be similarly expressed with the roles of "$\leq$" and "$\geq$" interchanged. For every linear inequality, $\mathbf{X}$, on n real-valued variables, let $\mathcal{S}(\mathbf{X})$ denote the set of points in the $n-$dimensional space $\Re^n$ that satisfy $\mathbf{X}$. The set of points satisfying a Boolean combination of linear inequalities is then inductively defined as follows: (i) $\mathcal{S}(\mathbf{X_1} \wedge \mathbf{X_2}) = \mathcal{S}(\mathbf{X_1}) \cap \mathcal{S}(\mathbf{X_2})$, (ii) $\mathcal{S}(\neg\mathbf{X_1}) = \Re^n \setminus \mathcal{S}(\mathbf{X_1})$, (iii) $\mathcal{S}(\mathbf{X_1} \vee \mathbf{X_2}) = \mathcal{S}(\mathbf{X_1}) \cup \mathcal{S}(\mathbf{X_2})$. We use $\top$ and $\perp$ to denote constraints for which the set of satisfying points is $\Re^n$ and $\emptyset$, respectively. It is easy to see that we can apply De Morgan's laws to Boolean combinations of linear inequalities, while remaining consistent with the definition of $\mathcal{S}(\mathbf{X})$. Hence, every such Boolean combination of linear inequalities can be reduced to *negation normal form*, where only individual linear inequalities are negated. However, every linear inequality can be negated by removing the negation operator and by replacing each"$<$" by "$\geq$", "$>$" by "$\leq$", and vice versa. In addition, the distributivity laws of Boolean logic allow every Boolean combination of linear inequalities to be expressed in a sum-of-products form. This gives us a normal form for expressing Boolean combinations of linear inequalities. In this form, every Boolean combination of linear inequalities, $\mathbf{X}$, has the form $\bigvee_j \mathbf{X_j}$, where each $\mathbf{X_j}$ is a conjunction of linear inequalities without the negation operator. We will henceforth consider all Boolean combinations of linear inequalities to be in this normal form unless stated otherwise. We will also use the terms "constraint" and "Boolean combination of linear inequalities" interchangeably.

Given an ATD, let $\mathbf{X_1}$ denote the conjunction of all assume constraints, and let $\mathbf{X_2}$ denote the conjunction of all guarantee constraints represented in the diagram. If $\mathcal{S}(\mathbf{X_1}) \subseteq \mathcal{S}(\mathbf{X_2})$, all guarantee constraints are satisfied whenever the assume constraints are satisfied. Since our premise is that the system satisfies all assume constraints, the interface can be declared free of timing problems with respect to the ATD if $\mathcal{S}(\mathbf{X_1}) \subseteq \mathcal{S}(\mathbf{X_2})$. If, however, $\mathcal{S}(\mathbf{X_1}) \not\subseteq \mathcal{S}(\mathbf{X_2})$, the system may violate the guarantee constraints. To prevent this from happening, we must find a set of additional constraints, $\mathbf{C_s}$, relating the circuit delays and clock periods, such that $\mathcal{S}(\mathbf{X_1} \wedge \mathbf{C_s}) \subseteq \mathcal{S}(\mathbf{X_2})$. $\mathbf{C_s}$ may be viewed as a set of *sufficient* delay constraints for ensuring that all guarantee constraints in the ATD are satisfied. A *necessary* delay constraint, $\mathbf{C_n}$, for correct operation of the circuit can similarly be viewed as one that satisfies $\mathcal{S}(\mathbf{X_1} \wedge \mathbf{X_2}) \subseteq \mathcal{S}(\mathbf{C_n})$. As an example, the set of assume constraints, $\mathbf{X_1}$, in Fig. 1(b) is given by $(t(\mathsf{c_s}) - t(\mathsf{g_s}) - \Delta_\mathsf{s} = 0) \wedge$

$(t(\mathsf{b_s})-t(\mathsf{c_s}) = 0) \wedge (t(\mathsf{a_r})-t(\mathsf{b_s})-\Delta^* = 0) \wedge (t(\mathsf{g_r^2})-t(\mathsf{g_r^1})-\mathsf{T_R} = 0) \wedge (t(\mathsf{c_r^1}) - t(\mathsf{g_r^1}) - \Delta_\mathsf{R} = 0) \wedge (t(\mathsf{c_r^2}) - t(\mathsf{g_r^2}) - \Delta_\mathsf{R} = 0) \wedge (t(\mathsf{g_r^1}) - t(\mathsf{g_s}) - \Delta_\mathsf{SR} = 0) \wedge (t(\mathsf{c_r^1}) - t(\mathsf{c_r^2}) < 0) \wedge (t(\mathsf{g_r^1}) - t(\mathsf{g_r^2}) < 0)$. Here, we have used constraints with the "=" operator as a shorthand for denoting a conjunction of constraints obtained by replacing the "=" operator with "$\leq$" and "$\geq$" operators. The set of guarantee constraints, $\mathbf{X_2}$, is given by $(t(\mathsf{a_r}) - t(\mathsf{c_r^1}) - \mathsf{H} \geq 0) \wedge (t(\mathsf{c_r^2}) - t(\mathsf{a_r}) - \mathsf{SU} \geq 0)$. It can be verified that $\mathcal{S}(\mathbf{X_1}) \not\subseteq \mathcal{S}(\mathbf{X_2})$. Therefore, the interface may malfunction, and we must derive a sufficient constraint, $\mathbf{C_s}$, on the set of ATD edge labels (symbolic circuit delays and clock periods), such that $\mathcal{S}(\mathbf{X_1} \wedge \mathbf{C_s}) \subseteq \mathcal{S}(\mathbf{X_2})$.

A trivial solution to the above problem is obtained by setting $\mathbf{C_s}$ to $\perp$. However, this solution is not useful since it does not help the designer build a correct interface. Since the set of constraints $\mathbf{X_1}$ is always assumed to be satisfied by the system, for $\mathbf{C_s}$ to be meaningful, we must ensure that $\mathcal{S}(\mathbf{C_s} \wedge \mathbf{X_1}) \neq \emptyset$. In order to achieve this, we partition the set of real-valued variables in $\mathbf{X_1}$ and $\mathbf{X_2}$ into two sets, $\mathcal{D}$ and $\mathcal{T}$. The set $\mathcal{D}$ contains all variables that appear as edge labels in the ATD, while $\mathcal{T}$ contains all other variables. We use the notation $\mathbf{X_i} \downarrow \mathcal{D}$ to denote the set of constraints obtained by projecting $\mathbf{X_i}$ on the variables in $\mathcal{D}$, i.e. all variables in $\mathcal{T}$ are eliminated. If $\mathbf{X_i}$ is expressed in the normal form described above, i.e., $\mathbf{X_i} = \bigvee_j \mathbf{X_{i,j}}$, then we have $(\bigvee_j \mathbf{X_{i,j}}) \downarrow \mathcal{D} = \bigvee_j (\mathbf{X_{i,j}} \downarrow \mathcal{D})$ since projection distributes over disjunction. Since every $\mathbf{X_{i,j}}$ is a conjunction of linear inequalities, each $\mathbf{X_{i,j}} \downarrow \mathcal{D}$ can be obtained by the classical Fourier-Motzkin elimination procedure [11]. We can now obtain necessary and sufficient constraints on the set of ATD edge delays for correct operation of the circuit as follows.

$$\mathbf{C_n} = (\mathbf{X_1} \wedge \mathbf{X_2}) \downarrow \mathcal{D} \tag{1}$$
$$\mathbf{C_s} = (\mathbf{X_1} \wedge \neg((\mathbf{X_1} \wedge \neg\mathbf{X_2}) \downarrow \mathcal{D})) \downarrow \mathcal{D} \tag{2}$$

In equation (2), $(\mathbf{X_1} \wedge \neg\mathbf{X_2}) \downarrow \mathcal{D}$ gives a set of necessary constraints on ATD edge delays for the system to satisfy $\mathbf{X_1}$ and violate $\mathbf{X_2}$. Its negation, therefore, gives sufficient constraints for the system to satisfy $\mathbf{X_2}$ whenever it satisfies $\mathbf{X_1}$. The conjunction with $\mathbf{X_1}$ weeds out spurious constraints that don't satisfy $\mathbf{X_1}$. Finally, projecting on the set, $\mathcal{D}$, of ATD edge delays gives the required set of sufficient constraints. Since $\mathcal{S}(\mathbf{X_1} \wedge \neg((\mathbf{X_1} \wedge \neg\mathbf{X_2}) \downarrow \mathcal{D}))$ is contained in $\mathcal{S}(\neg((\mathbf{X_1} \wedge \neg\mathbf{X_2}) \downarrow \mathcal{D}))$, it follows that $\mathcal{S}(\mathbf{C_s}) \subseteq \mathcal{S}(\neg((\mathbf{X_1} \wedge \neg\mathbf{X_2}) \downarrow \mathcal{D}))$. Note that equation (2) is semantically similar to the Presburger logic formulation of sufficient constraints in the work of Amon et al [1]. Note also that $\mathbf{C_s}$ obtained using the above method may indeed turn out to be $\perp$. In this case, we say that we are unable to derive sufficient constraints for correct operation of the circuit. A set of necessary constraints may, however, still be obtained by using equation (1). In general, $\mathcal{S}(\mathbf{C_s}) \subseteq \mathcal{S}(\mathbf{C_n})$. However, if $\mathcal{S}(\mathbf{C_s})$ equals $\mathcal{S}(\mathbf{C_n})$, we have a set of necessary and sufficient conditions for correct operation of the interface circuit. Using equations (1) and (2) in the

ATD of Fig. 1(b), we obtain the following necessary and sufficient constraints for correct operation of the interface in Fig. 1(a):

$$(\Delta^* \;\geq\; \Delta_{SR} + (\Delta_R - \Delta_S) + H) \;\wedge \qquad (3)$$

$$(\Delta^* \;\leq\; \Delta_{SR} + (\Delta_R - \Delta_S) + (T_R - SU)) \qquad (4)$$

The task of designing a correct interface now reduces to controlling the circuit delays such that inequalities (3) and (4) are satisfied. The symbols Δ_R, Δ_S, SU and H refer to known delays determined by clock buffering trees in the sender and receiver modules, and by flip-flop characteristics. If we assume that the interface in Fig. 1(a) does not modulate the receiver's clock (i.e., non-stoppable clocking), then T_R is also a known quantity. Thus, inequalities (3) and (4) can be met by ensuring that Δ^* tracks Δ_{SR}. If, in addition, Δ_{SR} is known (possibly due to a fixed relation between the sender and receiver clocks), inequalities (3) and (4) give upper and lower bounds for Δ^*. The interface designer is then simply required to ensure that the delay Δ^* of the interface circuit lies within these bounds.

3. Analyzing synchronization and sequencing constraints

We have seen above how timing constraints represented by ATDs can be analyzed to obtain necessary and sufficient delay constraints for correct operation of an interface circuit. However, such analysis is oblivious of the sequencing constraints arising from interfacing protocols and rational clock relations. Sequencing constraints enforce certain sequences of events and forbid other sequences from happening. Knowledge of sequencing restrictions can potentially allow us to derive less restrictive delay constraints for correct operation of an interface circuit. Consequently, it is advantageous to consider the interaction of sequencing and synchronization constraints when performing timing analysis. Unfortunately, earlier work in this area [1, 17, 30, 3, 6, 21, 12] require the user to determine all possible interactions of sequencing and synchronization constraints, and then capture each scenario explicitly using suitable diagrams. As discussed in Section 1, this approach has several drawbacks, especially in a design-analyze-redesign setting, where a designer may wish to mix and match several interfacing protocols, rational clock relations and synchronization constraints. In this section, we describe a technique for combining independently specified sequencing and synchronization constraints to derive delay constraints for correct operation of an interface circuit.

We will use an example GALS system, as shown in Fig. 2(a), to develop and illustrate our approach. In this system, there are three LS modules, M_1, M_2 and M_3, that communicate with each other. Each module M_i is clocked by CK_i with period T_i. To illustrate how information about rational clock

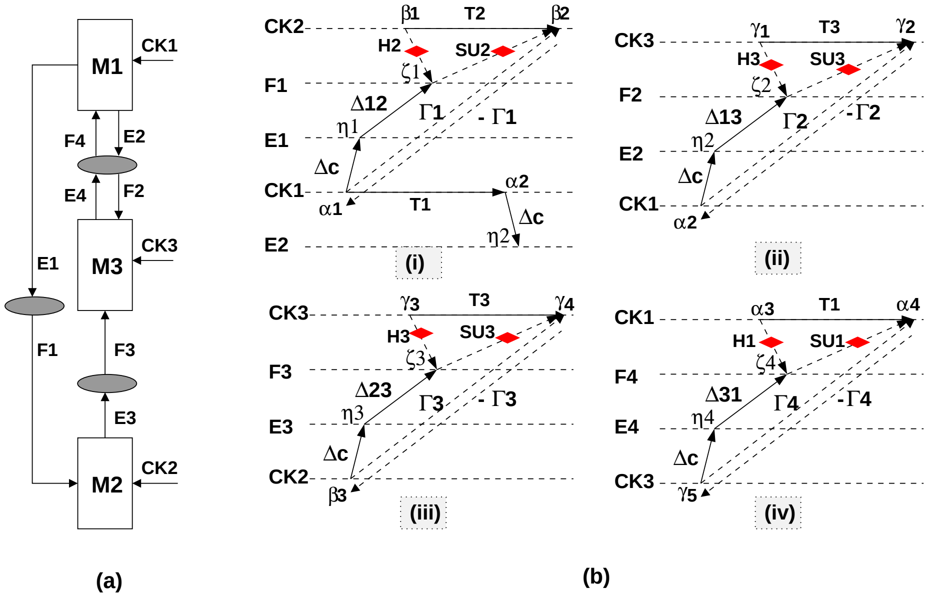

Figure 2. (a) An example GALS system, and (b) ATDs capturing interfacing scenarios

relations can be incorporated in our analysis, we assume that the three clocks are rationally related. Channels E_i and F_i are used for communication between the modules and the interfaces. A message sent on channel E_i is received by an interface circuit (shown shaded in Fig. 2(a)), which then forwards it to the destination module though channel F_i. Each module uses a specific protocol for communicating with the other modules. For ease of exposition, we assume the following simple protocol. Module M_1 first sends message e_1 to M_2, and then sends message e_2 to M_3 in the next local clock cycle. Module M_2, on receiving e_1, sends message e_3 to M_3 after a few local clock cycles. Once M_3 receives messages e_2 and e_3 (in no specific order), it sends message e_4 to M_1 in the next local clock cycle. The sequence of messages then repeats endlessly with an unbounded number of local clock cycles of M_1 occurring between the receipt of e_4 and the transmission of the next e_1. All messages are assumed to be generated synchronously with the clock ticks of the sending modules. In addition, we assume that all circuit and interconnect delays are small compared to the clock periods. Specifically, we make the following assumptions: (A1) all clock-buffering delays are small compared to the clock periods, (A2) the delay between a clock tick and a synchronously generated message e_i is uniformly Δ_c for all three modules, and (A3) for each synchronously generated message on E_i, the total delay from the clock tick that generates the message to the message being forwarded on $\mathsf{F_i}$ is small compared to the clock periods.

Fig. 2(b) shows four ATDs representing synchronization constraints in different interfacing scenarios for the above system. In this figure, all events are assumed to have AND causality (i.e., square vertices). Events on clock $\mathsf{CK_1}$ are labeled α_j, events on clock $\mathsf{CK_2}$ are labeled β_k and events on clock $\mathsf{CK_3}$ are labeled γ_m. Events on signals $\mathsf{E_i}$ and $\mathsf{F_i}$ are labeled η_i and ζ_i respectively. The delay introduced by the interface circuits in forwarding a message from $\mathsf{M_i}$ to $\mathsf{M_j}$ is denoted $\Delta_\mathsf{i,j}$, the setup and hold-times of module $\mathsf{M_i}$ are denoted $\mathsf{SU_i}$ and $\mathsf{H_i}$ respectively, and the delay from a sender's clock tick that triggers a message to the receiver's clock tick that samples the message is denoted Γ_p.

While the ATDs in Fig. 2(b) represent synchronization constraints between events, they do not provide information about sequencing constraints arising from the interfacing protocol and rational clock relations. Sequencing constraints between events are represented by an independently specified *sequence diagram*. Formally, let AP be a set of atomic propositions (used for labeling states), Evt be a set of event names (used for labeling edges), and $Clk \subseteq Evt$ be a set of clock event names. A sequence diagram is a $4-$tuple (Q, q_0, R, μ), where Q is a finite set of states, $q_0 \in Q$ is the initial state, $R \subseteq Q \times (2^{Evt} \setminus \emptyset) \times Q$ is a set of labeled directed edges, and $\mu : Q \rightarrow 2^{AP}$ labels each state with a subset of atomic propositions. Like in Kripke structures [10], every state in a sequence diagram is assumed to have at least one outgoing edge.

Fig. 3 shows a sequence diagram for the system depicted in Fig. 2(a). In this figure, edge labels denote abstract events, i.e., clock ticks and message events. Events on clock $\mathsf{CK_i}$ are represented as $\mathsf{C_i}$, while events on channel $\mathsf{F_i}$ are represented as $\mathsf{f_i}$. The set of clock events is $Clk = \{\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}\}$, and the set of all events is $Evt = \{\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}, \mathsf{f_1}, \mathsf{f_2}, \mathsf{f_3}, \mathsf{f_4}\}$. It is important to note that these labels need not be the same as the vertex labels used in the ATDs of Fig. 2(b). Note also that we have not represented events on channel $\mathsf{E_i}$ in the sequence diagram, although these events are present in the ATDs. This poses no problems since a sequence diagram is drawn independently of ATDs, and is not required to represent sequences of events on all signals in a system. In the system of Fig. 2(a), for example, when module $\mathsf{M_j}$ communicates with module $\mathsf{M_k}$ through the i^{th} interface, although each event on $\mathsf{F_i}$ is preceded by a corresponding event on $\mathsf{E_i}$, it is possible to represent the sequence of events on the clocks and $\mathsf{F_i}$'s without referring to the events on $\mathsf{E_i}$'s. This is because the total delay $\Delta_\mathsf{c} + \Delta_\mathsf{j,k}$ has been assumed to be smaller than any clock period (assumption (A3) above). An edge label containing more than one event, e.g. $\{\mathsf{C_1}, \mathsf{C_2}\}$, signifies that all the specified events occur *simultaneously* and cause the corresponding state transition to happen. This is useful for representing rational clock relations, where a pair of rationally related clocks must tick simultaneously once every few cycles. The sequence

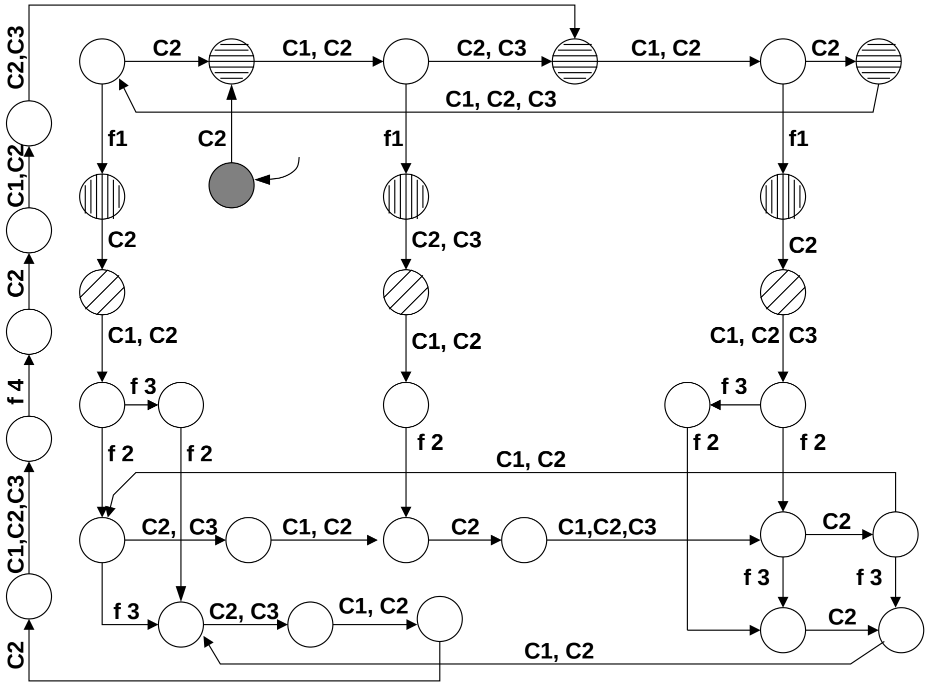

Figure 3. A sequence diagram for the system in Fig. 2(a)

diagram in Fig. 3 assumes that the clock periods of M1, M2 and M3 are in the ratio $2 : 1 : 3$. The states in Fig. 3 have been shown shaded in different ways to denote labeling with different sets of atomic propositions. Although each state is labeled with an appropriate set of atomic propositions, not all states have been shown shaded for clarity of presentation. The initial state has been indicated with a short incoming arrow.

A path π in a sequence diagram is a finite sequence of states $(s_0, s_1, \ldots, s_k)$ such that $(s_i, \sigma_i, s_{i+1}) \in R$ for some $\sigma_i \in 2^{Evt} \setminus \emptyset$, for all i in 0 through $k-1$. We denote the sequence of edge labels, $(\sigma_0, \ldots, \sigma_{k-1})$, in path π by $\mathsf{Seq}(\pi)$. Given a sequence diagram, the set of allowed event sequences is the set of event sequences obtainable from $\mathsf{Seq}(\pi)$ for paths π starting from the initial state. For example, the sequence diagram in Fig. 3 specifies that module M_3 sends message f_4 one local clock cycle after sampling the latest of messages f_2 and f_3. Similarly, it specifies that module M_2 sends message f_3 only in specific clock cycles after sampling message f_1. This could be attributed, for example, to the unavailability of port E_3 in the other cycles. Clearly, these sequencing aspects of the protocol are not captured (and need not be captured) by ATDs. Note that the assumptions about circuit and interconnect delays being small compared to the clock periods are also reflected in the sequence diagram. For example, when a synchronous event, say the sending of a message on channel E_1, is triggered by clock tick C_1, the sequence diagram specifies that the corresponding forwarded event, f_1, occurs before any subsequent tick of a rationally related clock. Thus, sequence diagrams are suited for specifying sequencing constraints without referring to details of synchronization constraints, while ATDs are suited for the complementary purpose.

Since a sequence diagram is a finite state description of allowable sequences of events, it can be constructed in one of several ways for constructing finite state transition systems. A compositional approach to building sequence diagrams of complex GALS systems from sequence diagrams of individual modules is described in [5]. This approach effectively computes the asynchronous parallel product of the sequence diagrams of individual modules and of rationally related clocks, and then filters out some state transitions from the resulting product based on suitable criteria. Details of this method are beyond the scope of this paper, and are available in [5]. For purposes of our discussion, however, we will assume that a sequence diagram describing sequencing constraints is available to us.

Having described sequence diagrams and ATDs, we now wish to combine the constraints represented in both these specification formalisms. The causal relations represented in an ATD give rise to a partial order of the events depicted in the ATD. However, not all linearizations of this partial order may be allowed by the sequencing constraints represented in a sequence diagram. Therefore, we must first identify those linearizations of the partial order that are allowed by the sequence diagram, and perform timing analysis only on these sequences of events. A naive approach to doing this would be to examine all sequences allowed by the sequence diagram, and to identify those subsequences that correspond to linearizations of the partial order represented by the ATD. However, this approach is not effective since a sequence diagram may indeed represent infinitely many event sequences. In order to address this problem, we introduce the notions of *path guards* and *anchor events*. As we will see shortly, these two notions provide a mechanism to zoom down on specific sequences of events that are allowed by a sequence diagram and are also consistent with the partial order represented by an ATD.

Formally, a path guard is a tuple (ϕ, m, n), where ϕ is an ECTL [10] formula on the set of atomic propositions (AP) used to label the states of the sequence diagram, m is a positive integer and n is a non-negative integer. An anchor event is a specially designated event (vertex) in an ATD. The signal corresponding to the anchor event is called the *anchor signal*. In combining the sequencing constraints represented by a sequence diagram with the synchronization constraints represented by an ATD, we will assume that events on the anchor signal are represented in the sequence diagram. Given a path guard $\mathcal{G} = (\phi, m, n)$, we define $\Pi(\mathcal{G})$ to be the set of all paths $(s_{-n},$

$\ldots, s_{-1}, s_0, s_1, \ldots, s_m)$ in the sequence diagram that satisfy the following two constraints: (i) state s_0 satisfies ϕ, and (ii) sub-path $(s_0, s_1, \ldots, s_m)$ is a prefix of an infinite path satisfying the path formula obtained by removing the outermost existential path quantifier in ϕ. We do not impose any additional constraints on the sub-path $(s_{-n}, \ldots, s_{-1})$. Thus, $(s_{-n}, \ldots, s_{-1}, s_0)$ is any sequence of $n+1$ states in the sequence diagram, ending in s_0. The "window of interest"defined by path guard $\mathcal{G}$ is the set of all edge-label sequences $\mathsf{Seq}(\pi)$, where $\pi \in \Pi(\mathcal{G})$. We will henceforth denote this set as $\mathsf{Win}(\mathcal{G})$. Since multiple edges in the sequence diagram may share the same label, a sequence in $\mathsf{Win}(\mathcal{G})$ may have multiple elements containing the same event name. This is undesirable when referring to elements by the collection of event names they represent. Hence, we rename the events in each element of each sequence of $\mathsf{Win}(\mathcal{G})$ by appending the index (in the sequence) of the element to the event names. We will henceforth assume that all elements in sequences of $\mathsf{Win}(\mathcal{G})$ have been renamed in this manner.

Let u and v denote the maximum number of outgoing and incoming transitions for each state in the sequence diagram, and let $States(\phi)$ denote the set of states that satisfy ϕ. If $\mathcal{G} = (\phi, m, n)$, the total number of sequences in $\mathsf{Win}(\mathcal{G})$ is bounded above by $|States(\phi)|.u^m.v^n$. In practice, however, the total number of sequences can be made much smaller by choosing ϕ, n and m appropriately. Given an ATD and a sequence diagram with a path guard $\mathcal{G}$, we check only those event sequences that are obtainable from sequences in $\mathsf{Win}(\mathcal{G})$ to see if they correspond to linearizations of the partial order represented by the ATD. Thus, path guards are used to prune the set of event sequences that are examined from the sequence diagram. Path guards are also useful in situations where an interfacing protocol may have different modes of operation, and in certain modes, synchronization failures of some (possibly dummy) messages are of no concern. In such cases, by choosing appropriate atomic propositions and path guards, we can effectively restrict the event sequences analyzed for synchronization failures to those where synchronization errors are indeed of concern. Our choice of ECTL in path guards is motivated by the fact that designers often describe an interfacing scenario by referring to a computation path with existential assertions about subpaths and states. The set of states that satisfy an ECTL formula ϕ can be easily computed using the explicit state CTL model checking algorithm [10]. Since this algorithm also labels all states with sub-formulas of ϕ, it is straightforward to find the sub-path $(s_0, \ldots s_m)$. Once $(s_0, \ldots s_m)$ is obtained, the path $(s_{-n}, \ldots, s_m)$ is obtained by simply going back n steps from s_0 in the sequence diagram.

Let $\sigma = (\sigma_{-n}, \ldots, \sigma_{m-1})$ be a sequence of edge-labels in $\mathsf{Win}(\mathcal{G})$. In order to determine whether an event sequence obtained from σ is consistent with a linearization of the partial order represented by an ATD, we first identify the set of signals, Θ, whose events are represented in both the ATD and the sequence diagram. We then simplify σ by projecting out (eliminating) all events on signals that are not in Θ. Let the simplified sequence be called σ'. The sequence σ is said to be consistent with a linearization of the partial order represented by the ATD iff there exists a mapping, λ, from ATD events on signals in Θ to events in σ', satisfying the following conditions:

C1: The anchor event in the ATD is mapped to the first event on the anchor signal in the subsequence $(\sigma_0, \sigma_1, \ldots, \sigma_{m-1})$.

C2: Each ATD event $\mathsf{e_i}$ on signal $\mathsf{x} \in \Theta$ is mapped to an event on x in σ'.

C3: If event $\mathsf{e_2}$ causally follows event $\mathsf{e_1}$ in the ATD, and if $\mathsf{e_1}$ is mapped to an event in σ_i and $\mathsf{e_2}$ is mapped to an event in σ_j, then $i < j$.

C4: If in addition to condition C3 above, event $\mathsf{e_2}$ is plotted to the immediate right of event $\mathsf{e_1}$ on the horizontal line corresponding to signal x in the ATD, then $\mathsf{e_1}$ and $\mathsf{e_2}$ denote consecutive events on x and hence, there must be no event on signal x in σ_k for all k satisfying $i < k < j$.

Because of constraints C2 and C4, once the leftmost event on a signal in an ATD is mapped to an event in σ', the mapping of the remaining ATD events on the same signal is uniquely determined. Specifically, there is only one way to map ATD events on the anchor signal due to constraints C1, C2 and C4. Thus, if the ATD represents events on r signals other than the anchor signal, and if there are k_i events on the i^{th} signal in the ATD, the number of possible mappings is bounded above by $\Pi_{i=1}^{r}(n+m-k_i+1)$. Conditions C2 and C3 further restrict this number significantly. In practice, the total number of mappings satisfying the above constraints is small, and these mappings can be obtained by explicit enumeration.

Given a sequence diagram and an ATD with a path guard $\mathcal{G}$ and an anchor event, let $\mathsf{ConsWin}(\mathcal{G})$ denote the set of sequences in $\mathsf{Win}(\mathcal{G})$ that are consistent with linearizations of the partial order represented by the ATD. For $\sigma \in \mathsf{ConsWin}(\mathcal{G})$ and $\mathsf{e} \in Evt$, let $\mathsf{Consec}(\mathsf{e}, \sigma)$ denote the set of all event-name pairs (e_i, e_j) such that e_i and e_j are consecutive events on signal e in the sequence σ. For each $\sigma \in \mathsf{ConsWin}(\mathcal{G})$, and for each mapping λ satisfying conditions C1 through C4 listed above, we now define $\mathbf{Constr}(\sigma, \lambda)$ to be the conjunction of the following constraints:

B1: $\bigwedge_{l=-n}^{m-1} \bigwedge_{e_i, e_j \in \sigma_l} \; t(e_i) - t(e_j) \; = \; 0$

B2: $\bigwedge_{l=-n}^{m-2} \bigwedge_{e_i \in \sigma_l} \bigwedge_{e_j \in \sigma_{l+1}} \; (t(e_i) - t(e_j) \; \leq \; 0)$

B3: $\bigwedge_{\mathsf{C} \in \mathit{Clk}} \bigwedge_{(e_i, e_j) \in \mathsf{Consec}(\mathsf{C}, \sigma)} \; (t(e_j) - t(e_i) - \mathsf{T_C} \; = \; 0)$, where $\mathsf{T_C}$ denotes the period of the clock corresponding to C

B4: $t(\mathsf{e}) = t(\lambda(\mathsf{e}))$, for each ATD event e on signals in Θ.

Constraint B1 is needed since edges of a sequence diagram may be labeled by sets containing multiple events, and hence σ_l may indeed be a set of simultaneous events. Constraint B2 captures the sequencing information between events, while B3 encodes the periodicity information of clocks. Finally, constraint B4 captures the mapping between ATD events and events in σ. Note that these constraints arise from the operation of the system, and hence are of the assume type. If $\mathbf{X_1}$ and $\mathbf{X_2}$ denote the set of assume and guarantee constraints represented by the ATD, the new set of assume constraints after incorporating the sequencing information is $\mathbf{X'_1} = \mathbf{X_1} \wedge \mathbf{Constr}(\sigma, \lambda)$. The set of guarantee constraints remains the same, while the set, $\mathcal{D}'$, of variables on which the constraints must be projected now includes desired ATD edge delays and all clock periods. Constraints for correct execution of an event sequence obtained from σ under the mapping λ can now be derived by replacing $\mathbf{X_1}$ with $\mathbf{X'_1}$ and $\mathcal{D}$ with $\mathcal{D}'$ in equations (1) and (2) of Section 2.1. Let us denote these constraints by $\mathsf{C_n}(\sigma, \lambda)$ and $\mathsf{C_s}(\sigma, \lambda)$, respectively. The overall constraints for correct operation of the interface are then obtained as: $\mathbf{C_s} = \Big(\bigwedge_{\sigma \in \mathsf{ConsWin}(\mathcal{G})} \bigwedge_{\lambda} \mathbf{C_s}(\sigma, \lambda)\Big)$, and $\mathbf{C_n} = \Big(\bigvee_{\sigma \in \mathsf{ConsWin}(\mathcal{G})} \bigvee_{\lambda} \mathbf{C_n}(\sigma, \lambda)\Big)$, where we consider only those mappings, λ, that satisfy conditions C1 through C4 listed above. If multiple ATDs are specified, $\mathbf{C_s}$ (respectively $\mathbf{C_n}$) obtained from each ATD must be conjoined to get the final sufficient (respectively necessary) constraints. The methodology described above has been implemented in an automatic tool that uses the Parma Polyhedra Library (PPL) [2] as a back-end engine for manipulating sets of convex polytopes representing conjunctions of linear inequalities.

To illustrate our approach on the example of Fig. 2(a), we use atomic propositions en_e_i, oc_f_i and sm_f_i ($1 \leq i \leq 4$) to label the states in the sequence diagram of Fig. 3. The proposition en_e_i ("enabling e_i") is used to label a state iff there is a clock tick labeling an outgoing transition from that state, such that this tick triggers message $\mathsf{e_i}$ (and hence $\mathsf{f_i}$) to be sent. Proposition oc_f_i ("f_i just occurred") is used to label a state iff the state is entered by a transition labeled $\mathsf{f_i}$. Similarly, proposition sm_f_i ("sampled f_i") is used to label a state iff there is a clock tick labeling an incoming transition to the state, such that this tick samples $\mathsf{f_i}$. For example, all states with vertical stripes in Fig. 3 are labeled oc_e_1, all states with horizontal stripes are labeled en_e_1, and all states with slanted stripes are labeled sm_e_1. The labeling of states with en_e_i, oc_f_i and sm_f_i for $2 \leq i \leq 4$ has not been shown in Fig. 3 for clarity of presentation.

In order to restrict the sequences of events that must be examined from the sequence diagram of Fig. 3, we use the path guards and anchor events as shown in Table 3. Note that the ECTL formula in each path guard refers to a computation path in which a clock tick triggers a message event, which is then forwarded through the interface and sampled by the receiver module.

Table I. Path guards, anchor events for ATDs of Fig. 2(b) and seq. diagram of Fig. 3

ATD of Fig. 2(b)	Path guard	Anchor event
(i)	$(en\_e_1 \wedge \mathrm{EF}(oc\_f_1 \wedge (\mathrm{EF}\ sm\_f_1) \wedge \mathrm{EF}(en\_e_2 \wedge \mathrm{EF}\ oc\_f_2)), 7, 0)$	ζ_1
(ii)	$(en\_e_2 \wedge \mathrm{EF}(oc\_f_2 \wedge \mathrm{EF}\ sm\_f_2), 6, 0)$	ζ_2
(iii)	$(en\_e_3 \wedge \mathrm{EF}(oc\_f_3 \wedge \mathrm{EF}\ sm\_f_3), 6, 0)$	ζ_3
(iv)	$(en\_e_4 \wedge \mathrm{EF}(oc\_f_4 \wedge \mathrm{EF}\ sm\_f_4), 6, 0)$	ζ_4

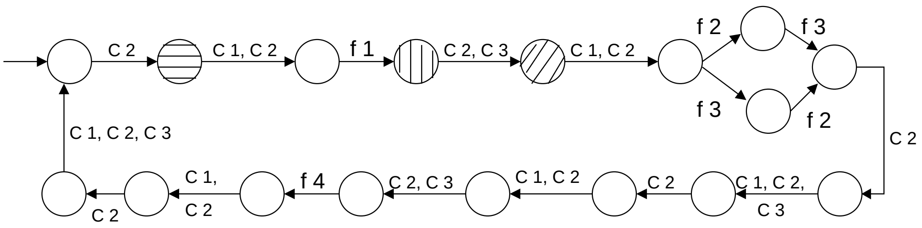

Figure 4. An alternative sequence diagram for the system in Fig. 2(a)

The "window sizes" (e.g., 6 going forward and 0 going backward) are chosen such that the event sequences in the window of interest are long enough to accommodate all events in the respective ATDs. Our knowledge of the relation between various clocks allows us to choose small window sizes in this example. In general, the window size must be chosen large enough to accommodate all events of interest starting from the state where the ECTL formula in the path guard evaluates to True. Thus, effective specification of path guards to prune the set of event sequences requires knowledge of the interfacing protocol and relations between clocks, in general. We expect this knowledge to be available to a GALS system designer in most cases. However, if the designer does not have adequate knowledge of the interfacing protocol, she may use conservative path guards like (True, m, n), where m and n are sufficiently large numbers. Unfortunately, the effectiveness of path guards in pruning the set of event sequences may be significantly reduced in such cases.

On analyzing the system shown in Fig. 2(a) using the ATDs of Fig. 2(b), the sequence diagram of Fig. 3 and the path guards and anchor events given in Table 3, we obtain a set of sufficient delay constraints for correct operation of the interface circuits. If we project these constraints on the clock periods, setup and hold-times, and interface circuit delays, and simplify the resulting inequalities, we get: $(\mathsf{T}_3 = 3.\mathsf{T}_2) \wedge (\mathsf{T}_1 = 2.\mathsf{T}_2) \wedge (\mathsf{H}_1 - \Delta_\mathsf{c} \leq \Delta_{31} \leq 2\mathsf{T}_2 - \mathsf{SU}_1 - \Delta_\mathsf{c}) \wedge (\mathsf{H}_2 - \Delta_\mathsf{c} \leq \Delta_{12} \leq \mathsf{T}_2 - \mathsf{SU}_2 - \Delta_\mathsf{c}) \wedge (\mathsf{H}_3 - \Delta_\mathsf{c} \leq \Delta_{13} \leq$

$\mathsf{T}_2 - \mathsf{SU}_3 - \Delta_c) \wedge (\mathsf{H}_3 - \Delta_c \leq \Delta_{23} \leq \mathsf{T}_2 - \mathsf{SU}_3 - \Delta_c)$. Necessary constraints for correct operation of the circuits can also be derived for this example using equation (1). However, we focus here on the sufficient constraints, since these are more interesting from an interface designer's perspective. Note that the upper bound on Δ_{31} obtained above is greater than the corresponding upper bounds for Δ_{12}, Δ_{13} and Δ_{23}, assuming all setup times to be equal. This allows more flexibility in the design of the interface between M_3 and M_1. The additional flexibility stems from the fact that the protocol represented in Fig. 3 allows an extra tick of CK_2 to occur between the sending of f_4 and its sampling by M_1.

To illustrate the advantage offered by our method when analyzing multiple combinations of sequencing and synchronization constraints, we now consider a different interfacing protocol for the system of Fig. 2(a), while keeping the synchronization constraints unchanged. Thus, we keep the ATDs of Fig. 2(b) unchanged, and represent the new protocol by the sequence diagram of Fig. 4. If we use the same path guards and anchor events as before, we now obtain the following sufficient constraints after simplification: $(\mathsf{T}_3 = 3.\mathsf{T}_2) \wedge (\mathsf{T}_1 = 2.\mathsf{T}_2) \wedge (\mathsf{H}_1 - \Delta_c - \mathsf{T}_2 \leq \Delta_{31} \leq \mathsf{T}_2 - \mathsf{SU}_1 - \Delta_c) \wedge (\mathsf{H}_2 - \Delta_c \leq \Delta_{12} \leq \mathsf{T}_2 - \mathsf{SU}_2 - \Delta_c) \wedge (\mathsf{H}_3 - \Delta_c - \mathsf{T}_2 \leq \Delta_{13} \leq 2\mathsf{T}_2 - \mathsf{SU}_3 - \Delta_c) \wedge (\mathsf{H}_3 - \Delta_c - \mathsf{T}_2 \leq \Delta_{23} \leq 2\mathsf{T}_2 - \mathsf{SU}_3 - \Delta_c)$. Note that these constraints are different from those obtained for the earlier protocol. This illustrates how the interaction of sequencing and synchronization constraints affects the conditions for correct operation of a circuit. This also demonstrates the convenience of specifying and analyzing multiple combinations of sequencing and synchronization constraints using our method. To illustrate the advantage of our approach over those that simply analyze the constraints represented in a timing diagram, we now apply the technique described in Section 2.1 to the ATDs represented in Fig. 2(b). This analysis along with knowledge of the rational relations between clocks gives us the following system of sufficient constraints for correct operation of the interfaces: $(\mathsf{T}_3 = 3.\mathsf{T}_2) \wedge (\mathsf{T}_1 = 2.\mathsf{T}_2) \wedge (\mathsf{H}_2 - \Delta_c + (\Gamma_1 - \mathsf{T}_2) \leq \Delta_{12} \leq \Gamma_1 - \mathsf{SU}_2 - \Delta_c) \wedge (\mathsf{H}_3 - \Delta_c + (\Gamma_2 - 3.\mathsf{T}_2) \leq \Delta_{13} \leq \Gamma_2 - \mathsf{SU}_3 - \Delta_c) \wedge (\mathsf{H}_3 - \Delta_c + (\Gamma_3 - 3.\mathsf{T}_2) \leq \Delta_{23} \leq \Gamma_3 - \mathsf{SU}_3 - \Delta_c) \wedge (\mathsf{H}_1 - \Delta_c + (\Gamma_4 - 2.\mathsf{T}_2) \leq \Delta_{31} \leq \Gamma_4 - \mathsf{SU}_1 - \Delta_c)$. Clearly, these constraints are not as useful to a designer as the constraints obtained using our analysis. This is expected, since the relations between Γ_i's and T_j's are not represented in the ATDs.

4. Comparison with related work

In Section 1, we briefly discussed how our work relates to earlier work on analysis of timing constraints represented using timing diagram formalisms.

In this section, we compare our work in greater detail with two earlier approaches that come closest to our work.

We first consider the work of Amon et al [1]. In their work, timing diagrams are used to represent symbolic timing constraints between events. Time is assumed to be discrete and the problem of finding constraints for correct operation of a system is reduced to that of eliminating quantifiers in a restricted class of Presburger formulae. The Omega libraries [22] for Presburger formulae manipulation are then used to obtain the desired constraints. Our formulation of sufficient constraints for correct operation of an interface circuit (see equation (2) in Section 2.1) semantically matches Amon et al's Presburger logic formulation if we assume time to be discrete. Our formulation, however, works for dense time as well. In addition, we provide a formulation for necessary delay constraints, which can be used to define an allowable design space, when a designer wishes to redesign parts of an interface circuit. Unlike the work of Amon et al, we use a normal form for Boolean combinations of linear inequalities. We also use PPL [2] for manipulating convex polytopes to derive constraints for correct operation of the system. Using a convex polytope manipulation library allows us to introduce safe approximations, wherever permissible, by replacing a union of convex polytopes with their convex hull, if the number of convex polytopes being represented becomes too large. For example, when computing $\mathbf{C_s}$ using equation (2), if $(\mathbf{X_1} \wedge \neg \mathbf{X_2})$ evaluates to the union of a large number of convex polytopes, the union can be safely approximated by the convex hull of the polytopes. The system of constraints, $\mathbf{C'_s}$, obtained with this approximation continues to be sufficient for ensuring correct operation of the interface circuit. Similarly, both $\mathbf{X_1}$ and $\mathbf{X_2}$ can be safely approximated by replacing unions with convex hulls when computing $\mathbf{C_n}$ using equation (1). More significantly, our method differs from that of Amon et al in that their approach does not allow the user to separately specify timing constraints and sequencing constraints between events. Since all constraints must be expressed using timing diagrams, the onus of determining all possible interactions of sequencing and synchronization constraints and representing them using timing diagrams rests on the user. This increases the chances of corner-case interaction scenarios being missed, in addition to making the user's task more tedious in a design-analyze-redesign setting, where different interfacing protocols, rational clock relations and synchronization constraints must be mixed and matched. In contrast, our method allows the designer to separately specify synchronization constraints and sequencing constraints arising from interfacing protocols and rational clock relations. Appropriate path guards can then be used to choose the right combination of sequencing and synchronization constraints for symbolic timing analysis. This separation of specifications, along with the ability to generate sequence diagrams of complex GALS systems in a compositional way [5], renders the job of analyzing different combinations of protocols, clock relations and synchronization constraints much easier. By automatically analyzing the interactions of sequencing and synchronization constraints, the chances of corner-case situations being missed is also eliminated.

The second work that comes close to our work is that of Clariso et al [9]. In their work, the behaviour of a circuit is represented using a finite state transition diagram, and symbolic relations between times of occurrence of events are built iteratively by systematically traversing the state transition diagram, starting from the initial state. While this gives a powerful technique for analyzing the timed behaviour of a circuit, it does not allow the user to focus the state space search and timing analysis to specific path fragments of interest. Instead, all paths starting from the initial state of the circuit must be explored to determine symbolic timing constraints. Thus, if a circuit has a large number of states (which is usually the case), Clariso et al's method necessarily generates a very large system of constraints in trying to traverse the entire state space. The representation and manipulation of such large sets of constraints poses severe bottlenecks, restricting the applicability of their method to small circuits. In contrast, the use of path guards in our approach allows the user to focus the symbolic timing analysis on short event sequences of interest in the sequence diagram. By employing ECTL model checking to identify these event sequences, we circumvent the need for representing and manipulating systems of constraints between symbolic delay variables when searching a large state space. Thus, our method restricts the computationally expensive step of symbolic timing analysis to short event sequences of interest defined by suitable path guards. This can lead to significant computational savings vis-a-vis the approach of Clariso et al [9]. Finally, Clariso et al's work does not allow a combined analysis of independently specified sequencing and synchronization constraints. In contrast, our solution allows symbolic delay constraints to be extracted by automatically combining independently specified sequencing and synchronization constraints. We believe this is a useful and important feature of timing analysis tools, especially in the era of SoC design using off-the-shelf IP cores.

5. Conclusion

Interface design and system integration will continue to be major challenges as we move into the era of large, high-performance SoCs built with off-the-shelf IP cores fed by multiple clocks. In order to enable designers to mix and match cores with different interfacing protocols, synchronization constraints and rational clock relations, it is important to equip them with tools and techniques that allow synchronization and sequencing constraints to be separately specified, and yet determine their combined effect on the behaviour of the system. In this paper, we have presented an approach for achieving this. The method presented here, coupled with the compositional approach for constructing sequence diagrams described in [5], provides a powerful tool in the hands of the interface designer. While our current implementation uses PPL as a back-end engine, we believe specialized engines can be designed for more efficient manipulation of boolean combinations of linear inequalities in which all coefficients range over a small set of non-negative integers (as is the case in our problem). This would directly translate to improved efficiency of our technique. While we have used path guards and anchor nodes to combine the constraints represented by sequence diagrams and those represented by ATDs, further research into alternative ways of specifying event sequences of interest is needed. Finally, it is worthwhile exploring other specification formalisms like delay-augmented net-charts [19] for describing sequencing and quantitative synchronization constraints, while still achieving a logical separation of the two specifications.

Acknowledgements

We are thankful to D. Gift Samuel and Supriya Garg for developing a prototype tool implementing the ideas presented in this paper. We are also grateful to the reviewers for their comments.

References

1. Amon, T., G. Borriello, T. Hu, and J. Liu: 1997, 'Symbolic Timing Verification of Timing Diagrams using Presburger Formulas'. In: *Proc. of ACM/IEEE Design Automation Conference*.
2. Bagnara, R. et al., 'PPL: The Parma Polyhedra Library'. http://www.cs.unipr.it/ppl/.
3. Burns, S. M.: 1991, 'Performance Analysis and Optimization of Asynchronous Circuits'. Ph.D. thesis, California Institute of Technology.
4. Chakraborty, A. and M. R. Greenstreet: 2003, 'Efficient Self-Timed Interfaces for Crossing Clock Domains'. In: *Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems*. pp. 78–88.
5. Chakraborty, S., 'Composing sequence diagrams for GALS systems'. http://www.cse.iitb.ac.in/∼supratik/tools/GALS_AutComp.
6. Chakraborty, S.: 1998, 'Polynomial-Time Techniques for Approximate Timing Analysis of Asynchronous Systems'. Ph.D. thesis, Stanford University.
7. Chaney, T. J. and C. E. Molnar: 1973, 'Anomalous Behavior of Synchronizer and Arbiter Circuits'. *IEEE Transactions on Computers* **C-22**(4), 421–422.
8. Chapiro, D. M.: 1984, 'Globally Asynchronous Locally Synchronous Systems'. Ph.D. thesis, Stanford University.
9. Clariso, R. and J. Cortadella: 2004, 'Verification of Timed Circuits with Symbolic Delays'. In: *Proc. of Asia and South Pacific Design Automation Conference*. pp. 226–231.
10. Clarke, E. M., O. Grumberg, and D. Peled: 1999, *Model Checking*. MIT Press.

11. Dantzig, G. B. and B. C. Eaves: 1973, 'Fourier-Motzkin Elimination and its Dual'. *Journal of Combinatorial Theory (A)* **13**, 288–297.
12. Fisler, K.: 1999, 'Timing Diagrams: Formalization and Algorithmic Verification'. *Journal of Logic, Language and Information* **8**(3), 323–361.
13. Ginosar, R. and R. Kol: 1998, 'Adaptive Synchronization'. In: *Proc. of Internanational Conf. Computer Design*. pp. 188–189.
14. Greenstreet, M. R.: 1993, 'STARI: A Technique for High-Bandwidth Communication'. Ph.D. thesis, Princeton University.
15. ITU-T: 1996, 'Message Sequence Charts (MSC), Recommendation Z.120'. Technical report.
16. Kessels, J., A. Peeters, P. Wielage, and S.-J. Kim: 2002, 'Clock Synchronization through Handshake Signalling'. In: *Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems*. pp. 59–68.
17. Khordoc, K. and E. Cerny: 1998, 'Semantics and Verification of Action Diagrams with Linear Timing Constraints'. *ACM Transactions on Design Automation of Electronic Systems* **3**(1), 21–50.
18. Mekie, J., S. Chakraborty, and D. K. Sharma: 2004, 'Evaluation of Pausible Clocking for Interfacing High-Speed IP Cores in GALS Systems'. In: *Proc. of International Conference on VLSI Design*. pp. 559–564.
19. Mekie, J., S. Chakraborty, G. Venkataramani, P. S. Thiagarajan, and D. K. Sharma: 2006, 'Interface Design for Rationally Clocked GALS Systems'. In: *Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems (to appear)*.
20. Muttersbach, J.: 2001, 'Globally Asynchronous Locally Synchronous Architechtures for VLSI Systems'. Ph.D. thesis, ETH Zurich.
21. Myers, C. J. and T. H.-Y.Meng: 2002, 'Synthesis of timed asynchronous circuits'. *IEEE Transactions on VLSI Systems* **1**(2), 106–119.
22. Pugh, W. et al., 'The Omega project'. http://www.cs.umd.edu/projects/omega.
23. Rosenberger, F. U., C. E. Molnar, T. J. Chaney, and T.-P. Fang: 1988, 'Q-Modules: Internally Clocked Delay-Insensitive Modules'. *IEEE Transactions on Computers* **C-37 (9)**, 1005–1018.
24. Roychoudhury, A., P. S. Thiagarajan, T.-A. Tran, and V. Zvereva: 2004, 'Automatic Generation of Protocol Converters from Scenario-based Specifications'. In: *Proc. of IEEE International Real-Time Systems Symposium*. pp. 447 – 458.
25. Seitz, C. L.: 1980, 'System Timing'. In: C. A. Mead and L. A. Conway (eds.): *Introduction to VLSI Systems*. Addison-Wesley, Chapt. 7.
26. Sjogren, A. E. and C. J. Myers: 1999, 'Interfacing Synchronous and Asynchronous Modules within a High-speed Pipeline'. *IEEE Transactions on VLSI Systems* **8**(5), 573–583.
27. Sproull, R. F. and I. E. Sutherland: 1985, *Stoppable Clock*. Sutherland, Sproull and Associates. Technical Memo 3438.
28. Stucki, M. J. and J. R. C. Jr.: 1979, 'Synchronization Strategies'. In: C. L. Seitz (ed.): *Proc. of the First Caltech Conference on Very Large Scale Integration*. pp. 375–393.
29. Weste, N. H. E. and K. Eshraghian: 1992, *Principles of CMOS VLSI design: A systems perspective*. Addison-Wesley Publishing Company.
30. Yen, T.-Y., A. Ishii, A. Casavant, and W. Wolf: 1998, 'Efficient Algorithms for Interface Timing Verification'. *Formal Methods in System Design* **12**(3), 241–265.
31. Yun, K. Y. and R. P. Donohue: 1996, 'Pausible Clocking: A First Step Toward Heterogeneous Systems'. In: *Proc. of International Conference on Computer Design*. pp. 118–123.