Assessing the Feasibility of Hardware Interface Designs in Microprocessor-based Systems

Marco A. Escalante Nikitas J. Dimopoulos

Technical Report ECE-95-1

Department of Electrical and Computer Engineering
University of Victoria, BC CANADA
P.O. Box 3055, Victoria, B.C., V8W 3P6

March 1995

Abstract

In this paper we address the feasibility of an "abstract" interface design. DAME is an expert microprocessor-based-systems designer. Once the system architecture has been selected and the major components (processors, memories, IO devices) have been instantiated from a component library, DAME designs the necessary glue logic to integrate the system. Such interface design is carried out according to the protocols followed by the components. The design is called "feasible" if it achieves the desired functionality and satisfies the timing constraints of the protocols. In this paper we address the problem of determining the feasibility of a design prior to its implementation. Because timing is an important aspect of a correct design, we use an interpreted timed Petri net to represent the timed behavior of protocols. Using a technique called timing analysis for synthesis we can check if a design is feasible even before logic synthesis is carried out.

1. Introduction

As the complexity of hardware systems increases, techniques that facilitate their design and verification are invaluable to hardware designers. The DAME project aims to automate the mechanical aspects of microprocessor-based system design [3]. DAME's main strength is its finer component representation down to the interfacing protocol level. DAME follows a top/down design process in which first a system architecture is decided, then the major components (processors, memories, and IO devices) are selected from a database according to user design constraints such as type of application, throughput, cost, etc. The next step is system integration, during which DAME designs the necessary glue logic. In this paper we address the problem of verifying that such interface design is *feasible*, i.e. the interface generates the necessary events at the expected times to accomplish the intended inter-component communication, before synthesis of the design is attempted. In this way not only the iteration of the design-synthesis-verification cycle can be broken but also an estimate of the quality of a design can be calculated (defined as how much delay margin a design possesses) which can be used to evaluate designs at a higher level of the design process.

Recent research in asynchronous design [8] indicates that interface design can benefit from a delay-insensitive design methodology which generates circuits that behave correctly even in the presence of variations on gate and wire delays. However it is not always possible to neglect timing information corresponding to either internal circuit delays or constraints on the environment for proper circuit operation [12]. This is particularly true in the design of microprocessor-based systems whose protocols are required to meet hard deadlines. For this purpose we use an interpreted timed Petri net which allows us to reason about operational *circuit delays* and environmental *timing constraints*.

Our model is suitable for symbolic timing analysis that finds bounds on the delays of the circuit to be synthesized before the actual circuit is implemented [4]. Delay-insensitivity is a special case of circuit design in which timing constraints are satisfied by any implementation regardless of the circuit delays. Moreover synchronous and partial handshake protocols can be considered as variations of the full handshake with missing event precedence links, requiring less control circuitry and exhibiting better performance at the expense of having to obey timing relationships for proper operation.

In the following section we survey related previous work. In section 3 a bus arbitration interface is used to motivate this work. The timed representation and the symbolic timing analysis is briefly presented in section 4. The formulation of the interface design as the *merging* of protocol graphs is discussed in section 5. Finally future directions are pointed out in the conclusions.

2. Related work

A microprocessor-based system is a collection of components which operate independently of one another but are required to communicate and synchronize with the rest of the system through communication structures called buses. The interface design problem arises during system integration when components are blended into a single entity. In general the design of an interface involves not only electrical and logical signal conditioning but also protocol conversion.

Signal transition graphs or STG's, a Petri net based representation formalism, have been used to describe the behavior of asynchronous control circuits [16, 1]. STG's were first applied to the design of delay-insensitive circuits which assumes unbounded wire and gate delays. Although a very powerful design concept, delay-insensitivity is not realistic for describing the behavior of microprocessor components.

Pioneering work by Nestor and Thomas [12] identified the necessity of dealing with timing constraints in the design of interfaces. Recently some work has been done in extending STG's to model circuit delays. Myers and Meng [11] used a conservative estimate of gate delays and available environmental timing constraints to remove links of the original STG specification of an asynchronous circuit that become redundant when timing bounds on the gate delays are taken into consideration. Their synthesis procedure relies on an algorithm that determines an upper bound on the maximum distance between two events in an acyclic graph. The general case, which involves solving a system of min/max inequalities, was shown in [10] to be NP-complete. An algorithm that finds exact bounds on the maximum distance between two events was reported in [6] which can only analyze systems with max terms.

During the integration phase of the design, which takes place before synthesis, the path delays of the interface are still unknown. None of the aforementioned approaches can handle unknown delays directly, i.e. without having to assume values for the unknown delays and iteratively checking that the assumed values satisfy the given timing constraints. In [4] we proposed a symbolic timing analysis that finds tightest bounds on variable (unknown) delays using the available information in the form of circuit delays and timing constraints. In this paper we show that such technique can be used as the backbone of a procedure that can determine the feasibility of an interface design without requiring to generate first an implementation of the interface. The importance of this method is that it allows the designers to evaluate timing aspects of an abstract design at a higher level of the design process.

3. Motivation: A bus arbitration interface design

Interface logic is necessary to interconnect the components that comprise a system. Microprocessor components transfer information in the form of signals through wires that interconnect their ports. According to the functionality, signals are grouped into different interfacing capabilities (e.g., data transfer, interrupt, bus arbitration). The interfacing protocol enforces the correct exchange of information by defining the ordering and timing of elementary operations or *actions* [2]. Signal transitions are used to encode the actions of the protocol. In general the components that constitute the system may use different protocols. One of the main tasks of the interface is to perform protocol conversion.

Consider for example a microprocessor system which consists of multiple masters shown in Figure 1. A master is able to initiate a data transfer via a shared resource: the data transfer bus. The bus arbitration lines are used to guarantee that at most one master requesting the bus takes over the DTB at any given time. A bus arbitration protocol is defined by the standard bus, and each of the masters may use a different bus arbitration protocol. The interface circuit between a master and the standard bus generates the input actions to both components.

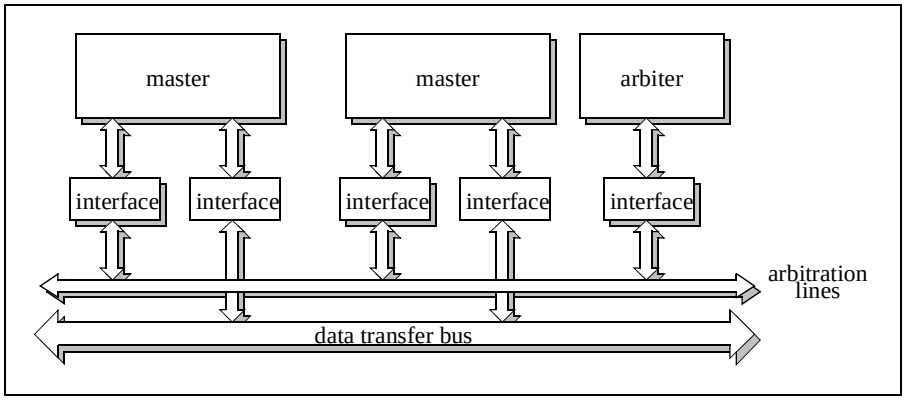

Figure 1. Multi-master system.

The timing diagram of a typical bus arbitration protocol used by a master is shown in Figure 2. The master signals to the arbiter that it wants to use the bus by asserting the output REQ signal (a suffix '*' in a signal name indicates that the signal uses negative logic, i.e. it is asserted low), and it waits for ACK to become asserted before seizing the bus. When the master ends its transaction, it releases REQ. The arbiter will then release ACK so that another arbitration cycle can take place.

In Figure 2 input signals are shown underlined. The REQ/ACK signals are control signals which can be asserted and negated, while the DTB is a group of lines that are initially in a high-impedance state, and are driven by the master when it utilizes the bus. Control signal

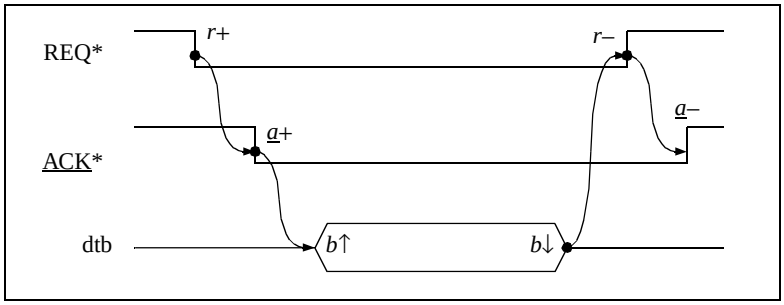

Figure 2. Master bus arbitration protocol.

transitions change the state from negated to asserted or viceversa, denoted respectively by +/–, while the data bus lines can switch from disabled to enabled or viceversa, denoted respectively by ↑/↓.

The Petri net shown in Figure 3a describes the protocol of Figure 2. Places in the net model delays, and transitions of the net represent signal changes. There are two different types of timed behavior: circuit delays and environmental constraints. A circuit delay represents the response time of the master and is modeled by operational places connected by continuous links. For instance, in Figure 3 place p_3 describes the internal delay in the master from receiving the bus grant ($\underline{a}$+) to driving the DTB lines. Timing constraints specify how the external circuit should behave for proper operation. This is shown by constraint places connected by dashed links. For example, place p_2 in Figure 3 specifies that a grant (a+) is expected to occur some time after the master issues a request (r+). The net in Figure 3a is a marked graph (i.e., every place in the net has one input transition and one output transition). Marked graphs can be represented more succinctly as synchronization graphs [15] in which transitions are drawn as nodes and places are drawn as links as shown in Figure 3b.

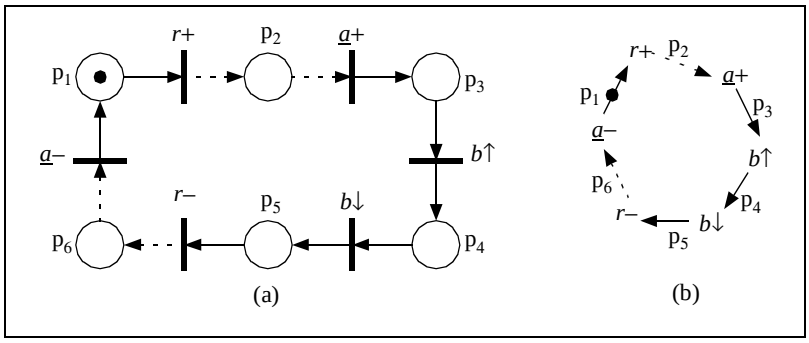

Figure 3. Master bus arbitration protocol: (a) Petri net; (b) synchronization graph.

From the example above one can see that a protocol specification contains not only the component's internal operation but also the allowed behavior of the environment. It is the interface's task to generate an appropriate environment for the master. Figure 4 shows a bus arbitration interface between a DMA device (master) and the VMEbus. In the structural

view the connectivity of input and output signals can be identified. The DMA device has a pair of control signals, while the VMEbus protocol uses three control signals and one status signal (BUSFREE).

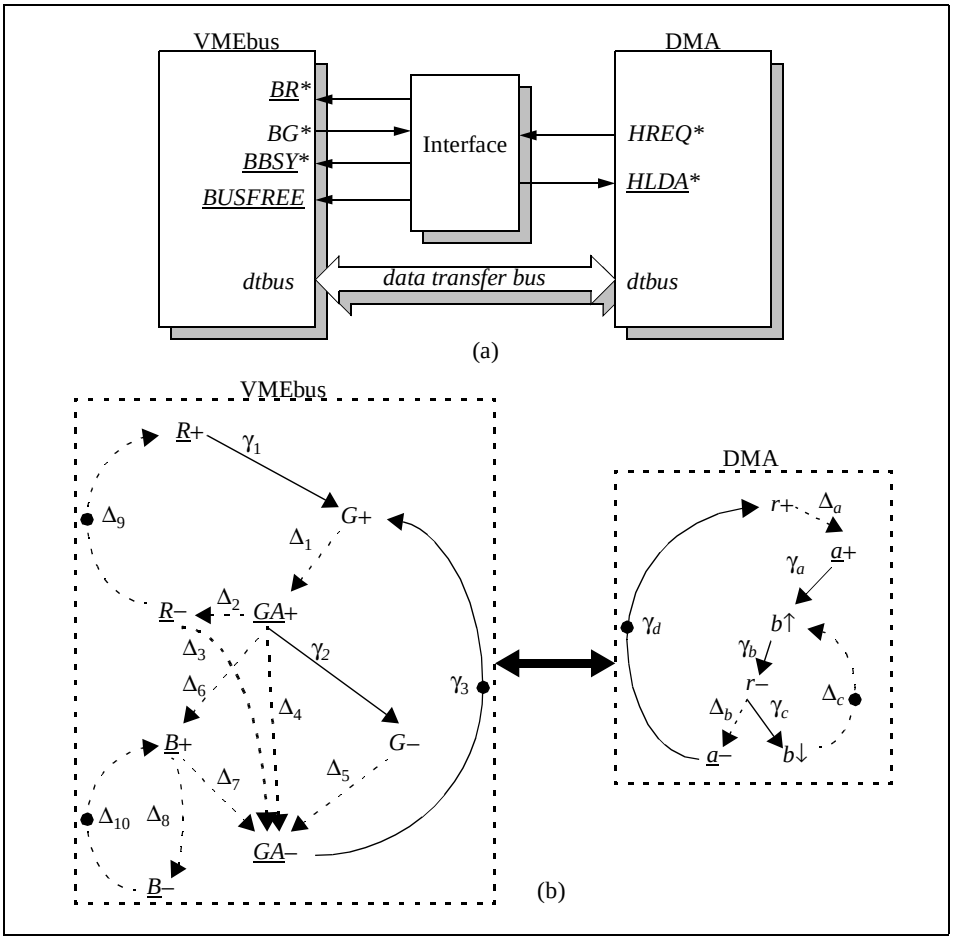

Figure 4. Bus arbitration interface: (a) structural view; (b) behavioral view.

The corresponding protocols are shown in Figure 4b. In the protocol graphs, the following shorthands are used for the names of the signals: HREQ is *r*, HLDA is *a*, the data transfer bus is *b*, BR is *R*, BG is *G*, BBSY is *GA* and BUSFREE is *B*. The DMA device uses a variation of the fully handshaken protocol shown in Figure 3. The difference is that the request signal is released before completion of the bus transaction. This fact has important repercussions in the design. In both protocols the release of the request triggers a sequence of events that eventually will grant the bus to a master. However while in the fully handshaken protocol the release of request occurs when the bus is already available, in the partial handshake (shown in Figure 4b right) potentially another master can take over the bus before the previous master relinquishes the lines if delay γ_c is large compared to the other delays. Therefore in the latter protocol it is necessary to check that such hazard never occurs. Constraint Δ_c states that the bus lines should be released before a new cycle starts.

Although it can be thought that the fully handshake is a more robust protocol, this comes at a price. The partial handshake is faster because it allows the arbitration to take place (the path from $r-$ to $a+$) at the same time as the transaction is completed ($b\downarrow$).

The VMEbus bus arbitration protocol is more involved. It defines the arbiter's behavior. After a request is received ($R+$) a grant is generated ($G+$). A master being granted the bus acknowledges the grant by asserting the grant-acknowledge signal ($GA+$) to which the arbiter responds by releasing grant ($G-$). GA is used to speed up the arbitration similarly to the partial handshake by allowing the arbitration to take place while the last part of the transaction is still in progress. The master must monitor the availability of the bus (B) before driving the bus lines.

The interface reads the output events of both protocols and generates the necessary input events. In Figure 4 the components to be interconnected follow different protocols so that protocol conversion is required. It is not possible to check if the constraint links are satisfied using conventional verification techniques because they require that the interface delays δ be known. Instead we find the tightest bounds on the delays δ such that the timing constraints are satisfied. If there is no such set of values then the interface must be redesigned (e.g. by modifying some or all of the δ links in the merged graph or by choosing different components). Otherwise those bounds can be used to select an appropriate target technology and guide time-driven synthesis tools. Finally a correct realization of the interface must not exceed the bounds computed by the analysis.

In the following section we discuss the formal tools that lead to a symbolic timing analysis for a subclass of Petri nets, namely the subclass that can represent AND and OR causality [18].

4. Timed representation of protocols

In the following subsection we introduce our timed STG representation. Then our symbolic timing analysis is posed as a transposition of the constraint satisfaction problem, namely given a set of known operational delays and timing constraints, determine possible values of unknown interface path delays. In microprocessor-based system design, the known operational delays and timing constraints correspond respectively to circuit delays and timing constraints specified in the component data sheets, while the unknown path delays are the delays of the interface logic that is yet to be synthesized.

4.1 Timed Petri net model

A *timed* Petri Net is a quintuple $TPN = \langle P, T, F, M_0, \Lambda \rangle$ where P is a non-empty set of places, T is a non-empty set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, $M: P \rightarrow N$ is the marking function, and $\Lambda: P \rightarrow I+$ is the time labeling function that assigns to each place a non-negative compact interval $\lambda \in I+$. N is the set of the natural numbers and $I+$ is the set of non-negative compact real intervals.

The set of places is partitioned into two subsets P_o and P_c. Time labels assigned to places belonging to P_o, the set of *operational places*, are used to model circuit delay. Time labels assigned to places belonging to P_c, the set of *constraint places*, are used to specify required behavior of the environment for proper operation of the circuit. The flow function is naturally partitioned by the sets P_o and P_c, i.e., $F = F_o \cup F_c$ where $F_o \subseteq (P_o \times T) \cup (T \times P_o)$ and $F_c \subseteq (P_c \times T) \cup (T \times P_c)$. The preset (postset) of a transition t is the set of incoming places to (outgoing places from) t and is denoted $\bullet t$ ($t\bullet$). The intersection of $\bullet t$ ($t\bullet$) with P_o is denoted as $\bullet t_o$ ($t_o\bullet$), likewise for $\bullet t_c$ ($t_c\bullet$). The firing rule of the Petri net is extended accordingly to take into account the different behavior of operational and constraint places.

Firing rule:

1. A transition t is enabled when every place $p \in \bullet t_o$ contains a token.

2. An enabled transition fires immediately. When it fires, the transition sends tokens to every place $p \in t\bullet$ and anti-tokens to every place $p \in \bullet t$.

3. An operational place p labelled with $\lambda_p = [\tau_{min}, \tau_{max}]$ upon receiving a token at time τ makes it visible to transitions $t \in p\bullet$ at time $\tau + \tau_x$, where $\tau_x \in \lambda_p$. The token is held by the place until it is annihilated by an anti-token.

4. A constraint place p labelled with $\lambda_p = [\tau_{min}, \tau_{max}]$ upon receiving a token at time τ holds it during the interval $[\tau + \tau_{min}, \tau + \tau_{max}]$. If the constraint place receives an anti-token when it does not hold a token, it flags a constraint violation.

The use of anti-tokens is our mechanism of assigning to the places the responsibility of flagging violations.

4.2 Timed signal transition graphs

Ports are designated by unique names. Input port names are written underlined, e.g., $\underline{a}$, $\underline{b}$, $\underline{c}$, while output port names are written as a, b, c. Signals carry the values of ports through wires. Let X be the set of input signals and Z the set of non-input signals of a circuit. The set of signals is $Y = X \cup Z$. The set of signal transitions (or actions) is $A = Y \times \{+, -, \uparrow, \downarrow\}$.

A pair (a, +), written as a+, represents a transition of signal a from negated to asserted. Likewise the other transition symbols represent signal transitions from asserted to negated, from disabled to enabled, and from enabled to disabled respectively.

STG's are Petri nets whose transitions are interpreted as signal transitions. A timed STG is the triplet $\langle TPN, Y, L\rangle$ where TPN is a marked timed Petri net, Y is a set of signals, and $L: T \rightarrow A$ is a labelling function which assigns transitions $t \in T$ of the Petri net to signal transitions $a \in A$.

Not every interpretation of a Petri net describes a correct behavior of a circuit (e.g., if two successive transitions of the Petri net are labelled with the same signal transition). The *validity* of an STG can be checked by ensuring that the corresponding state graph is consistent [17]. The validity of timed STG's is further discussed in section 5.1. Several synthesis techniques for STG's have been reported in the literature [8, 11, 17]. The following definition states under which circumstances the time behavior of an STG is said to be *time-consistent.*

Definition 4.2.1.- A timed STG is time-consistent if no constraint place flags a violation during any possible execution of the STG.

In the following subsection we develop the analytical machinery to determine if a timed STG is time-consistent.

4.3 Symbolic timing analysis

In this subsection we formulate the time-consistency of periodic timed STG's as an optimization problem that avoids the enumeration of all possible executions. We use interval arithmetic to compute the time of occurrence of transitions due to operational places.

Let I be the set of real compact intervals. An interval operation $\otimes$ for $\alpha, \beta \in I$ is defined by:

$$\alpha \otimes \beta = \{a \otimes b : a \in \alpha \wedge b \in \beta\}$$

In particular expressions for interval addition, subtraction, and *min* and *max* functions are given by:

$$\alpha + \beta = [a_{min} + b_{min},\ a_{max} + b_{max}]$$

$$\alpha - \beta = [a_{min} - b_{max},\ a_{max} - b_{min}]$$

$$min\ (\alpha, \beta) = [min\ (a_{min}, b_{min}),\ min\ (a_{max}, b_{max})]$$

$$max\ (\alpha, \beta) = [max\ (a_{min}, b_{min}),\ max\ (a_{max}, b_{max})]$$

where $\alpha = [a_{min}, a_{max}]$ and $\beta = [b_{min}, b_{max}]$.

Consider the Petri net subclass of marked graphs ($|\bullet p| = |p\bullet| = 1$, and thus places can be drawn as links between two transitions). Transition *d* in Figure 5 has three incoming operational places shown as links labelled with intervals γ_i, i=1..3. The occurrence times of transitions *a*, *b* and *c* are also shown in Figure 5. The firing rule states that transition *d* sees a token in each of its incoming places at any time during the corresponding shadowed interval, and *d* is enabled when all three tokens on the incoming places are made visible to *d*. This occurs within the interval *max* ($\tau_a + \gamma_1$, $\tau_b + \gamma_2$, $\tau_c + \gamma_3$).

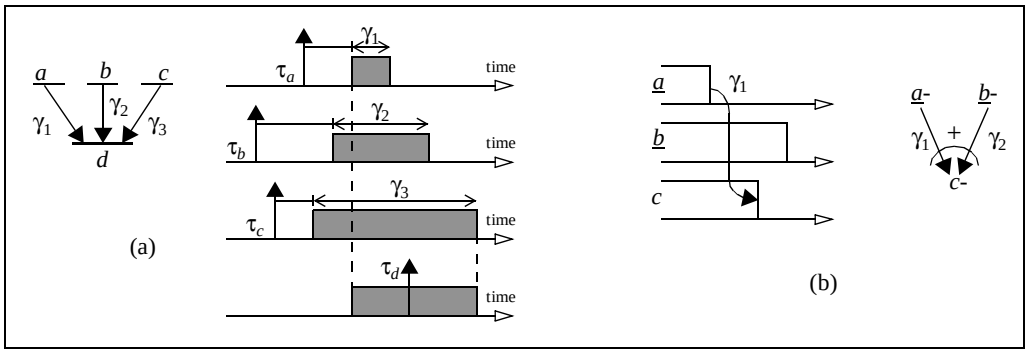

Figure 5. Firing of a transition: (a) AND causality; (b) OR causality.

A constraint place between two transitions *a* and *b* (see Figure 6) signals a violation *iff* τ_b does not occur within the constraint interval after the occurrence of τ_a. A constraint place is said to be time-consistent if it does not signal a violation under any possible execution of the STG. Let $\tau_x^{\,i}$ denote the time of the *ith* occurrence of transition *x*. To determine if the place ever signals a violation under any possible execution of the STG, one has to check the bounds on the time separation from $\tau_b^{\,i}$ to $\tau_a^{\,i}$, written $\tau_b^{\,i} - \tau_a^{\,i}$.

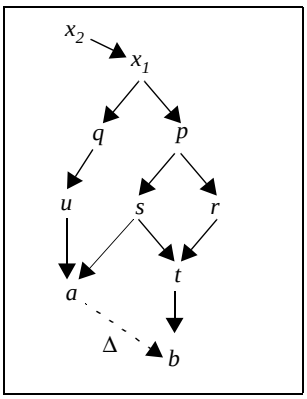

Figure 6. Fork transition for constraint Δ.

Definition 4.3.1.- A constraint place is time-consistent if for all occurrences *i*:

$$\tau_b^{\,i} - \tau_a^{\,i} \subseteq \Delta \qquad \text{[Eq. 1]}$$

Proposition 4.3.2.- An STG is time-consistent *iff* all its constraint places are time-consistent.

Proof.- It follows from Definition 4.2.1.

To compute the time interval difference in Eq. 1, we unfold the STG starting from the initial marking. The resulting unfolded graph is acyclic and infinite. Figure 7 shows a simple protocol between two signals and its corresponding unfolded graph. In our application we require that the execution of a protocol graph result in periodic behavior. Thus, after a finite transient, Eq. 1 becomes independent of the occurrence *i*. A common ancestor of both transitions *a* and *b* from which $\tau_b - \tau_a$ can be computed is called a *fork transition*.

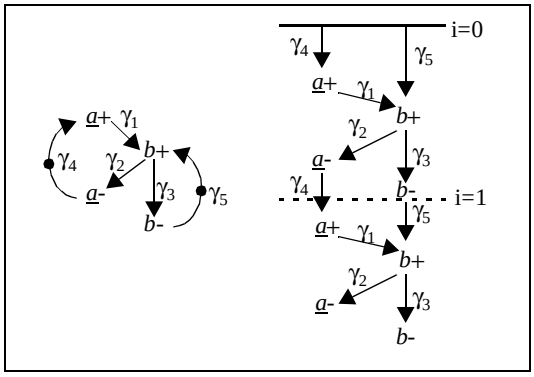

Figure 7. Simple signal transition graph and a partial view of its unfolded infinite acyclic graph.

Definition 4.3.3.- A transition *x* is called a fork transition for constraint Δ from *a* to *b* if there exist two lattices in the unfolded graph whose common least upper bound is *x*, and with greatest lower bounds *a* and *b* respectively such that, for every node in each lattice except *x*, all its ancestors belong to the corresponding lattice.

For example the fork transition for Δ in Figure 6 is x_1. Note that *s* does not qualify as a fork transition because *r*, which is an ancestor of *t*, does not belong to the lattice from *s* to *b*. The fork transition is not necessarily unique: x_2 in Figure 6 is also a fork transition for Δ. However the choice of fork transition is immaterial for the evaluation of Eq. 1.

After a fork transition *x* has been identified, the time separation is computed as the interval difference between the occurrence times of transitions *b* and *a* in the unfolded graph relative to *x*. For example the separation between transitions $b+^i$ and $\underline{a}+^i$ in Figure 7 for any cycle $i > 0$ (the first cycle corresponds to i=0) is $\{max\ (\gamma_2 + \gamma_4 + \gamma_1, \gamma_3 + \gamma_5)\} - \{\gamma_2 + \gamma_4\}$. The fork transition of $b+^i$ and $\underline{a}+^i$ is $b+^{i-1}$.

A causal constraint is a constraint place labeled [0, ∞). The following proposition is useful to simplify the timing analysis.

Proposition 4.3.4.- Let Δ be a causal constraint from *a* to *b*. If *a* is a fork transition for Δ, then Δ is always satisfied.

Proof.- If *a* is a fork transition, then $\tau_b - \tau_a$ can be computed by traversing a lattice starting from *a*, that yields a non-negative interval. It follows that if Δ is causal, it is always satisfied.

Eq. 1 involves the subtraction of interval expressions, each possibly containing *max* terms. Thus Eq. 1 is a nonlinear interval expression. Using an approach adapted from [10], it is possible to solve the constraint satisfaction problem by solving first a finite set of subproblems. A subproblem is produced by choosing a winner for each of the *max* terms. The solution of each subproblem can be formulated as a linear program which finds the minimum and maximum values of a linear interval expression (i.e., with the *max* terms removed) subject to the γ_i intervals and to the conditions imposed by the choices of winners in the *max* terms, which are also linear expressions on γ_i. The solution of the original problem is the union of the solutions of all subproblems. For notational clarity, in the sequel we denote intervals with Greek letters (e.g., γ, Δ) and a particular value within the interval with the Latin alphabet (e.g., $c \in \gamma$).

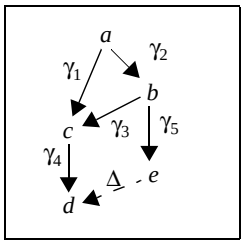

Figure 8. Constraint satisfaction.

Consider for example the graph shown in Figure 8. The constraint satisfaction equation is $\tau_d - \tau_e \subseteq \Delta$, where $\tau_d = max\,(\gamma_1, \gamma_2 + \gamma_3) + \gamma_4$, and $\tau_e = \gamma_2 + \gamma_5$ (*a* is the fork transition). There are two possible choices of winners for the unique *max* term. The subproblem obtained by choosing $\gamma_2 + \gamma_3 \geq \gamma_1$ generates the following linear program:

$$\text{min/max } \{c_3 + c_4\} - \{c_5\}$$

subject to

$$c_i \in \gamma_i,\ i = 1..5$$

$$c_1 - c_2 - c_3 \leq 0$$

where the conditions $c_i \in \gamma_i$ can be expanded into the conjunction of inequalities $c_i \leq \gamma_{i,max}$ and $-c_i \leq -\gamma_{i,min}$. For $\gamma_1 = [0, 90]$, $\gamma_2 = [0, 100]$, and $\gamma_3 = \gamma_4 = \gamma_5 = [10, 20]$, the solution of $\tau_d - \tau_e$ is [0, 30]. Similarly for the subproblem $\gamma_1 \geq \gamma_2 + \gamma_3$, $\tau_d - \tau_e = [0, 100]$. Thus for any Δ such that $[0, 100] \subseteq \Delta$, the constraint place of Figure 8 is time-consistent.

We now state the symbolic timing analysis formulation. Suppose that some of the operational intervals are unknown, denoted by δ_i. The constraint equations are now written in terms of known γ_i's, unknown δ_j's, and constraint Δ_k's. As before we can construct linear subproblems of the constraint satisfaction problem corresponding to a particular winner choice for each *max* term. For a given subproblem, a value y_k that satisfies the left-hand side of a constraint equation for Δ_k (i.e., $y_k \in \tau_b - \tau_a$) can be written as $y_k = f_b(c_i, d_j) - f_a(c_i, d_j)$, where f_a and f_b are two linear functions on the c_i's and d_j's such that $c_i \in \gamma_i$ and $d_j \in \delta_j$. According to Eq. 1, $y_k \in \Delta_k$. Possible values for the δ_j's must satisfy the following conditions:

$y_k \in \Delta_k$, k= 1..L,

$c_i \in \gamma_i$, i = 1..M,

$d_j \geq 0$, j = 1..N, and

conditions given by the choice of max terms.

where L is the number of constraint Δ_k's, M is the number of known operational γ_i's, and N is the number of unknown δ_j's.

The above conditions for a particular subproblem describe a set of feasible points $\{c_i, d_j\}$ which, when non-empty, is delimited by a (possibly unbounded) convex polytope. A convex polytope is the convex hull of its vertices, thus finding a finite number of vertices suffices to characterize a particular solution set (if the polytope is unbounded, only additional direction vectors describing the edges to infinity are required). Let *poly* = $\{c_i, d_j\}$ be the union of all the polytopes generated by the particular solutions. The total solution is the largest set $\{d_j^*\}$ such that $\{c_i, d_j^*\} \in$ *poly* for all values $c_i \in \gamma_i$.

Transition d in Figure 5a fires when all places in $\bullet d$ make a token visible. This is the standard AND causality in Petri nets. A complementary behavior, called OR causality in [18] can be described as follows: a transition fires as soon as one of the incoming places to the transition makes the token visible. (In [7] the terms AND/OR causality are referred as latest/earliest timing relationships respectively.) This is depicted in Figure 5b. Transition c- occurs as soon as the first of $\underline{a}$- or $\underline{b}$- occurs. This happens within the interval *min* ($\tau_{\underline{a}\text{-}} + \gamma_1$, $\tau_{\underline{b}\text{-}} + \gamma_2$). The OR behavior can be represented using standard Petri net constructs [18]. Our method can also handle nets with OR causality by replacing the max terms with min terms (a choice of winner in a min term generates a condition which is also linear). Note that marked graphs is a proper subset of the class of nets that can describe AND and OR causality.

Consider the circuit implementation of a D-element shown in Figure 9 which was reported in [6]. The D-element synchronizes two components that use handshakes to communicate. The left handshake $\underline{li}+ \rightarrow lo+ \rightarrow \underline{li}- \rightarrow lo-$ is interspersed with the right handshake

ro+→<u>*ri*</u>+→*ro*−→<u>*ri*</u>− as described by the STG shown in Figure 10a. A state variable *x* is used to differentiate two half cycles. Martin [9] uses the D-element to implement sequencing between two processes via the handshakes. Observe that the left handshake is passive, i.e., it is initially in a waiting state. Both the AND gate with inverted inputs and the buffer outside the D-element simulate its environment by generating the desired acknowledgment transitions after a gate delay.

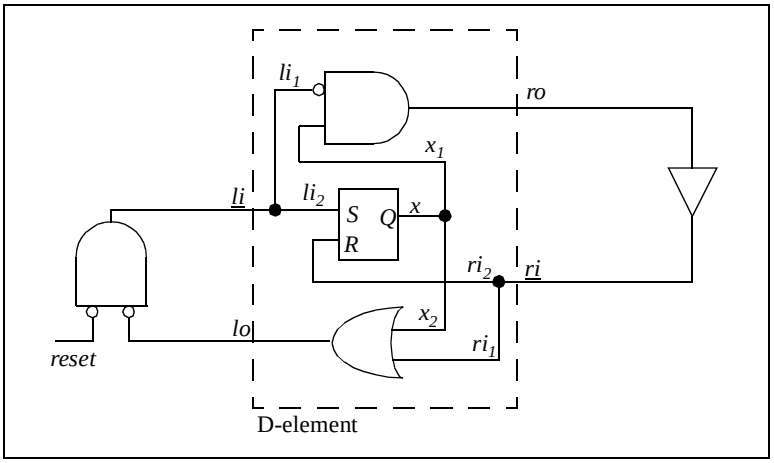

Figure 9. Circuit implementation of the D-element.

Figure 10b shows in detail the sequence of transitions in one cycle of the D-element. Operational links represent as usual the behavior of the circuit. Delays through gates are labelled with γ_i, and to distinguish wire delays, they are labelled with ω_i. The wire delays labelled with α and β and the constraint links have a special meaning as it will be clear shortly. Assume that the S-R flip-flop and all signals are initially set at zero. After a reset pulse, the first <u>*li*</u>+ transition is generated. That transition switches the S-R flip-flop to one, which in turn causes transition *lo*+ to occur. After the reset pulse the AND gate behaves as an inverter and so it generates <u>*li*</u>−. Now the AND gate of the D-element causes transition *ro*+, which is propagated to <u>*ri*</u>+. The flip-flop is reset, which subsequently produces the sequence *ro*−→<u>*ri*</u>−→*lo*−. If a transition is propagated through different paths to different parts of the circuit, new transitions are created to take into consideration that the paths may have different delays. For example, transition <u>*li*</u>+ forks transitions <u>li_1</u>+ and <u>li_2</u>+ to represent its arrival at the inverted input of the AND gate and the *S* input of the flip-flop respectively.

In the circuit implementation, malfunction may occur due to differences in the path delays of signals *ri*, *li*, and *x* to different parts of the circuit. For example, if transition li_1+ at the inverted input of the AND gate occurs after it has been propagated to x_1+, an undesirable glitch will appear at the output of the gate. In order to avoid these hazards, Martin [9] suggested to assume isochronic forks, i.e., that the delays of forked transitions created

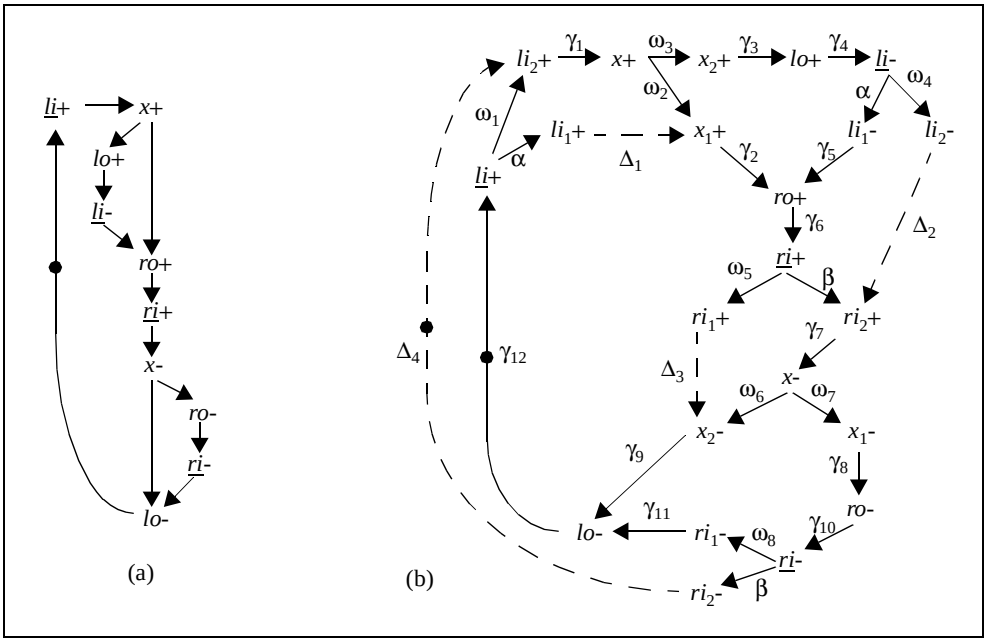

Figure 10. Behavior of the D-element: (a) abstract behavior; (b) detailed behavior showing the fork transitions.

from a common transition that branches out into different paths are negligible compared to other delays and thus the forked transitions will occur at *about* the same time. The hazard discussed above is precluded by the isochronic fork assumption.

Hulgaard *et al* observed in [6] that the isochronic fork assumption is too strong, and can be relaxed as follows: the circuit will function correctly as long as forked transitions that do not have a successor transition in the STG (and thus are not acknowledged) occur before they are used later in the execution of the circuit. This can be accomplished by adding causal Δ_i constraints from such fork transitions to the appropriate transitions. For example, li_1+ must occur before x_1+ arrives at the non-inverted input of the upper AND gate, otherwise the gate will produce a spurious pulse at *ro*; this is monitored by constraint Δ_1. The problem is to determine under which conditions the added constraints are satisfied.

Suppose that all gate delay ranges are $\gamma_i = [2, 3]$ and that all wire delay ranges are $\omega_i = [0, 1]$ with the exception of α and β which are to be determined. Hulgaard's procedure can check the constraints only for known values of α and β, so he does find ranges for α and β that satisfy the constraints by trial and error. It is not clear that all possible ranges for α and β can be found using such procedure.

Our symbolic timing analysis on the other hand finds all possible values for α and β that satisfy the constraints without iterations. First we write the four constraint equations corresponding to each Δ_i. For example the equation for constraint Δ_2 using *x*+ as the fork transition is written as follows:

$$\{max(\omega_2 + \gamma_2, \omega_3 + \gamma_3 + \gamma_4 + \gamma_5 + \alpha) + \gamma_6 + \beta\} - \{\omega_3 + \omega_4 + \gamma_3 + \gamma_4\} \subseteq \Delta_2 \quad \text{[Eq. 2]}$$

Note that it is always the case that $\omega_2 + \gamma_2 < \omega_3 + \gamma_3 + \gamma_4 + \gamma_5 + \alpha$. Thus Eq. 2 can be reduced to:

$$\{2\gamma + \alpha + \beta\} - \{\omega\} \subseteq \Delta_2$$

where we have dropped the subscripts of the operational labels. Likewise the other constraint equations are: $\{2\omega + \gamma\} - \{\alpha\} \subseteq \Delta_1$, $\{\beta + \gamma + \omega\} - \{\omega\} \subseteq \Delta_3$, and $\{2\omega + 2\gamma\} - \{\beta\} \subseteq \Delta_4$. The result of the timing analysis proves that all the constraints are satisfied if $\alpha = [0, 2]$ and $\beta = [0, 4]$. The solution is the shaded area shown in Figure 11. Note that the circuit will function properly even if β violates the isochronic fork assumption.

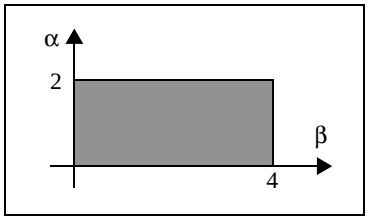

Figure 11. Solution polytope for delays α and β.

Another example is a fully-handshake data read transaction between a CPU and a RAM device (see Figure 12). A dual-rail encoding of the data signals [13] is used so that the accept action (*ack*+) can be generated after the data is received. If the four Δ-constraints are causal, it is evident from Proposition 4.3.4 that all the constraints are satisfied by the operational links. In this example, the solution polytope $\{\delta_1, \delta_2\}$ consists of the complete positive quadrant.

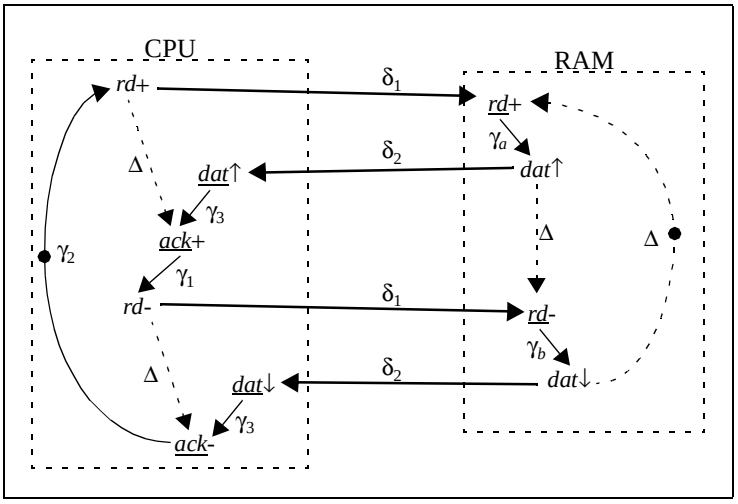

Figure 12. Fully handshake protocols.

5. Feasibility of the interface design

The interface design conceptualization is facilitated by an appropriate timed framework such as the one discussed in section 4. In a timed STG, operational links describe the internal operation of components while constraint links specify the desired environment. In this section we develop a test to determine if an interface design is feasible, that is, it produces a correct environment for the components to be interconnected. The test involves checking that the constraints are satisfied. Because no silicon has been assigned to the interface at this stage, values for the interface operational delays are not known. Therefore a symbolic timing analysis procedure is essential to perform a test for feasibility.

The starting point is to characterize what constitutes a valid specification. As mentioned before, a timed STG that describes the interfacing protocol of a component captures not only the internal operation of the device but also the expected behavior of the environment. Because the protocols that we are interested in are reactive, we also require that the STG be live and safe. To design the interface, we construct a *merged* graph which consists of the original protocol graphs with additional operational links that constitute the interface. There are some restrictions regarding the addition of new operational links. For instance, interface links cannot be drawn to output transitions of the protocol graphs which are generated internally by the components and are therefore inaccessible to the interface logic. Finally to guarantee that the purpose of the protocols is accomplished, semantic constraints must also be satisfied.

5.1 Valid specification

The significance of a valid specification is that it describes a correct behavior considering both the internal operation and the environment of a component. A valid specification is checked on the time reduction of a timed STG.

Definition 5.1.1.- A time reduction of a timed STG $TS = \langle TPN, Y, \Delta \rangle$ where $TPN = \langle P, T, F, M_0, \Lambda \rangle$, is the untimed STG $S = \langle PN, Y, \Delta \rangle$, where $PN = \langle P, T, F, M_0 \rangle$. Furthermore there is no partition defined in the place set of PN.

In the time reduction of a timed STG, the time labels are removed from the original graph and places are not partitioned into constraint and operational subsets.

Definition 5.1.2.- Let $S = \langle PN, Y, \Delta \rangle$ be a timed STG. S is said to be a valid specification if its time reduction has the following properties:

1. There is at least one simple cycle containing both transitions *a!* and *a!**.

2. In every simple cycle containing both transitions *a!* and *a!**, the transitions alternate.

3. There is one and only one token in every simple cycle of the graph.

The above properties reflect the fact that the protocols we are concerned with exhibit cyclic behavior. Condition 1 assumes return-to-zero cycles. Condition 2 guarantees the consistency of the graph. Condition 3 characterizes a live and safe graph.

5.2 Interface design and STG feasibility

A correct interface implements the expected environment in both protocol graphs by generating the necessary input transitions. Note that input/output transitions in the protocol specifications are output/input transitions of the interface. In the simplest case, where the operational behavior of one specification emulates the environment of the other, the interface reduces to wiring up the corresponding output/input pairs of transitions. In the general case, the interface may need to perform protocol conversion. There are some restrictions for the addition of operational interface links: it is not allowed to add any operational links to output transitions of the protocol graphs (output transitions are generated by the internal circuitry of the components and cannot be modified by the interface), and transitions on bundled lines (e.g., the data transfer bus lines) cannot be used to generate control events.

A semantic specification is a valid STG containing selected transitions of the specifications which are joined only by constraint links.

Definition 5.2.1.- Let TS_1 and TS_2 be two specifications with transition sets T_1 and T_2, and labeling functions Δ_1 and Δ_2. Let $T' \subseteq T_1 \cup T_2$ and Δ' be the labeling function that maps transitions of T' to the same signal transitions as given by Δ_1 and Δ_2. A semantic specification of TS_1 and TS_2 is a valid timed STG $TS' = \langle TPN', Y', \Delta' \rangle$ with $TPN' = \langle P', T', F', M_0', \Lambda' \rangle$ where all places are constraint places.

The semantic specification is meant to specify the goal to be achieved by exercising the protocols [14]. For example, Figure 13a shows the semantic specification for a bus arbitration cycle. In words, it specifies that once the data transfer bus is seized by a master (b+) the transaction must terminate (b–) before the bus can be taken over again.

Definition 5.2.2.- Given two valid specifications of two protocols together with their associated semantic specification, a complete STG is a timed STG $TS' = \langle TPN', Y', \Delta' \rangle$ such that:

1. The STG's of the protocol and semantic specifications are subgraphs of the complete STG.

2. Neither should interface operational links sink to output transitions of the protocol specifications nor connect a transition of a set of bundled signals to a transition of a control signal.

3. For every constraint in the complete STG there is a fork transition.

A complete STG describes the interface design. Condition 1 ensures that the protocol specifications (internal behavior plus environment) as well as the semantic functionality are taken into consideration for the interface design. Condition 2 forbids certain operational links. Condition 3 makes sure that the complete graph can be checked for constraint satisfaction. We now state conditions under which a given interface design is feasible. For this effect we shall use the symbolic timing analysis procedure discussed in section 4.3.

Definition 5.2.3.- A complete STG is called feasible if it is time-consistent.

In a time-consistent STG all timing constraints are satisfied. We emphasize that timing constraints in our framework not only specify timing relations between transitions but, more importantly, they define the environment of a component. In this sense, checking that the timing constraints of the complete graph are satisfied is equivalent to guaranteeing that the environment of the components is properly generated by the interface.

It is possible that several designs for a given interface are feasible. Currently we are investigating knowledge-based techniques to efficiently find feasible designs given the protocols and the semantic specification. In the following example we show how different interface designs can be measured by comparing their solutions of the symbolic timing analysis.

5.3 Bus arbitration interface example

A design representing the bus arbitration interface presented in section 3 (see Figure 4) is shown in Figure 13. The semantic specification (Figure 13a) specifies that once a transaction commences, it must finish before the next transaction may take place. The *complete* STG representing the interface design is shown in Figure 13b. One can recognize the two component protocols as subgraphs of the merged graph. New links (places) have been added to input transitions that correspond to the interface paths. Such links are labeled with δ indicating that their values are unknown at this moment; however they represent circuit delays (like the γ links). The added interface links are compliant with condition 2 of Definition 5.2.2. For instance, interface path delay δ_1 corresponds to the logic and wiring path that passes a request issued by the master to the arbiter.

Note that the timing relationship between the use of the bus (*b*) and the bus-free status signal (*B*) is inverted. One would expect that transitions *B*+ and *B*– should frame the utili-

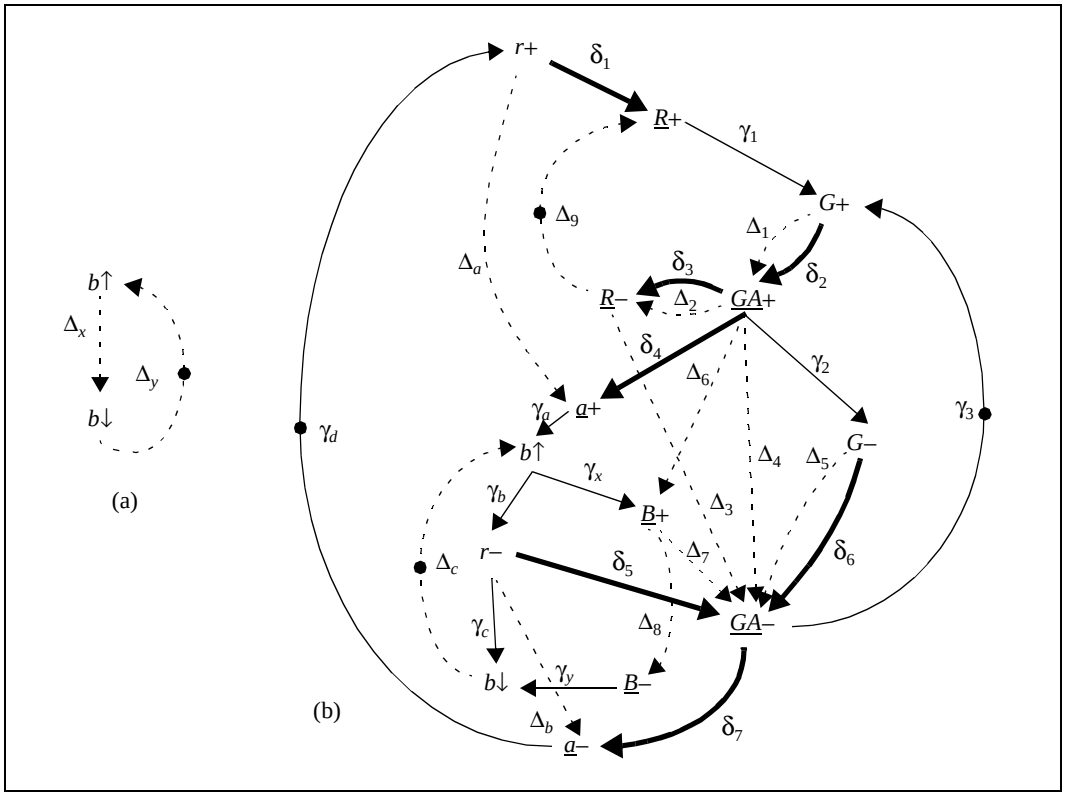

Figure 13. Bus arbitration controller: (a) semantic specification; (b) interface design.

zation of the bus (between $b\uparrow$ and $b\downarrow$) as shown in Figure 14a. However VMEbus allows the designer to use the address strobe signal, which belongs to the DTB, as indicator of the status of the bus and thus observing the relationship shown in Figure 14b. Constraint link Δ_c monitor the possibility of a bus collision. All constraint places Δ_i are causal (describing a precedence requirement) except Δ_3 and Δ_4, which are labeled with intervals [30, ∞) and [90, ∞) respectively. Let us investigate the effect of these two constraints on the unknown interface δ delays.

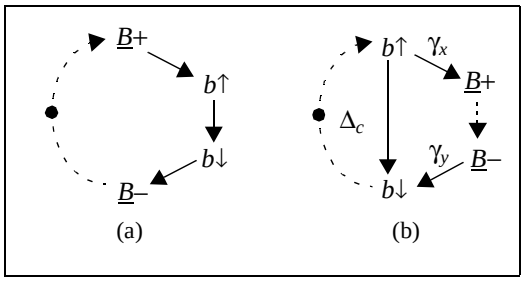

Figure 14. Bus busy status signal: (a) strobe relation; (b) actual relation.

First we write the constraint equations (see Eq. 1) for Δ_3 and Δ_4. The fork transition for both constraints is transition *GA*+. The constraint equations are given by the following expressions:

$$\{\max(\delta_4 + \delta_5 + \gamma_a + \gamma_b, \delta_6 + \gamma_2)\} - \{\delta_3\} \subseteq [30, \infty)$$

$$\{\max(\delta_4 + \delta_5 + \gamma_a + \gamma_b, \delta_6 + \gamma_2)\} \subseteq [90, \infty)$$

We proceed to apply the symbolic timing analysis procedure discussed in the previous section. We linearize the *max* term (common to both equations) by considering two cases:

1. $\delta_4 + \delta_5 + \gamma_a + \gamma_b \geq \delta_6 + \gamma_2$

$$\{\delta_4 + \delta_5 + \gamma_a + \gamma_b\} - \{\delta_3\} \subseteq [30, \infty)$$

$$\{\delta_4 + \delta_5 + \gamma_a + \gamma_b\} \subseteq [90, \infty)$$

2. $\delta_4 + \delta_5 + \gamma_a + \gamma_b \leq \delta_6 + \gamma_2$

$$\{\delta_6 + \gamma_2\} - \{\delta_3\} \subseteq [30, \infty)$$

$$\{\delta_6 + \gamma_2\} \subseteq [90, \infty)$$

Using the following values for the known delays: γ_2 = [15, 30], γ_a = [20, 80], and γ_b = [40, 100], one can write the following two sets of linear inequalities:

Case 1.	**Case 2.**
$-c_a - c_b + d_3 - d_4 - d_5 \leq -30$	$-c_2 + d_3 - d_6 \leq -30$
$-c_a - c_b - d_4 - d_5 \leq -90$	$-c_2 - d_6 \leq -90$
$15 \leq c_2 \leq 30$	$15 \leq c_2 \leq 30$
$20 \leq c_a \leq 80$	$20 \leq c_a \leq 80$
$40 \leq c_b \leq 100$	$40 \leq c_b \leq 100$
$c_2 - c_a - c_b - d_4 - d_5 + d_6 \leq 0$	$-c_2 + c_a + c_b + d_4 + d_5 - d_6 \leq 0$
$d_i \geq 0,\ i = 3..6$	$d_i \geq 0,\ i = 3..6$

Figure 15a shows of the solution polytope for δ_3, δ_4, δ_5, δ_6 (one of the axis is labeled $\delta_4 + \delta_5$ to display the solution in three dimensions). The polytope is the volume bounded by planes that extend to infinity in the directions shown by the five pointers, reflecting the fact that arbitrary large delays are accommodated by the handshakes in the protocols. Small values for the interface delays however can cause violations of the timing constraints. The relation between the δ's is shown in the polytope. For instance, the plane below the pointer starting at (δ_3, δ_4+δ_5, δ_6) = (30, 60, 0) is the region where the path through δ_4 and δ_5 is too fast with respect to the δ_3 path, which causes a violation of Δ_3. Informally a delay-insensitive circuit is defined as a circuit whose correct operation is independent of circuit delays. The bus arbitration interface is clearly not delay-insensitive, otherwise its solution would consist of the whole positive octant.

Consider now a slightly different scenario: interface path δ_6 is removed from the merged graph in Figure 13b. This new design is also feasible and its solution polytope is shown in Figure 15b. To compare both designs, we form the projection of the solution pol-

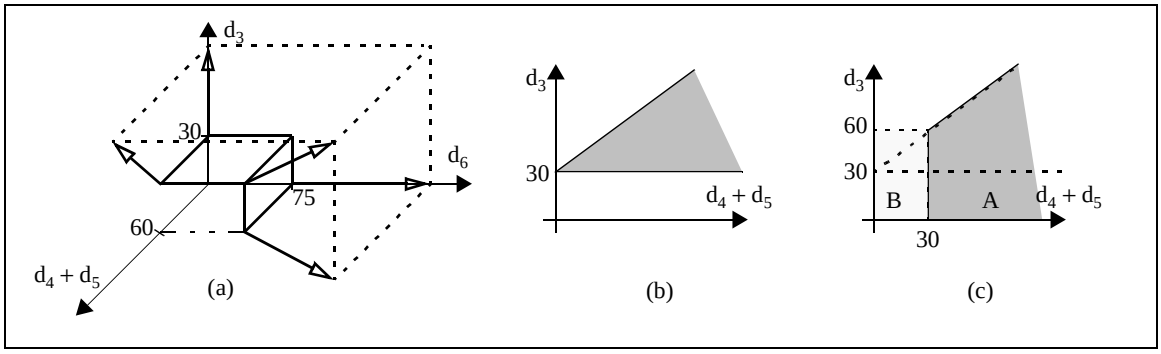

Figure 15. Solution polytope for $\Delta_3 = [30, \infty)$ and $\Delta_4 = [90, \infty)$: (a) With interface path δ_6; (b) without interface path δ_6; (c) projection of part (a).

ytope in Figure 15a into the (d_3, d_4+d_5) plane, which is shown in Figure 15c. Region A depicts the permitted values for the delays d_3, d_4, and d_5 for any value $d_6 \geq 0$. The first design has clearly a larger delay margin. Moreover if $d_6 \geq 75$, then region B is added to the projection of the solution, thus including completely the second design.

The information provided by the solution polytope can be advantageously used during synthesis, for instance to guide the layout and routing tools or, once the final delays are calculated in the final implemented circuit, to check that the final implementation complies with the interface design. The solution polytope can also be used to compare interface designs.

6. Conclusions

DAME, an expert microprocessor-based-systems designer, represents components at a finer detail, the component protocol, so that during system integration it can design the required interface circuitry. Our representation, based on a timed Petri net model, allows designers to reason about circuit delays and timing constraints that are necessary to describe accurately the protocols used by microprocessor components. In this paper we present how such a design is produced, by *merging* the protocol graphs of the components to be interconnected. Moreover, we state conditions under which the design is feasible, that is, it achieves its purpose (described by a semantic specification), and produces a correct environment for the interconnected components (described by the timing constraints specified in the protocols). Semantic and protocol specifications are represented uniformly in DAME's framework as timed signal transition graphs (STG's). By using a symbolic timing analysis procedure that finds tight bounds on unknown path delays, the interface design can be proven feasible before an implementation is carried out, thus avoiding the expensive iteration between design and synthesis.

Acknowledgments

The first author wants to thank Karim Khordoc and Professor Ed Cerny of the University of Montreal for an invaluable exchange of ideas regarding the modeling of hardware interfaces that took place during the author's workterm at BNR (Ottawa), and to Allan Silburt of BNR for making this interaction possible.

References

[1] T.-A. Chu, "On the models for designing *VLSI* asynchronous digital systems," *INTEGRATION, the VLSI journal*, no. 4, pp. 99–113, 1986.

[2] D. del Corso, H. Kirmann, and J. D. Nicoud, *Microcomputer buses and links*. Academic Press, 1986.

[3] N. J. Dimopoulos, K. F. Li, and E. G. Manning, "DAME: a rule-base designer of microprocessor-based systems", in *Proc. of the 3rd. International Conference of Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 716–725, July 1990.

[4] M. A. Escalante and N. J. Dimopoulos, "Timing analysis for synthesis in microprocessor interface design," in *Proceedings of the Seventh High-Level Synthesis Symposium*, pp. 23–28, May. 1994.

[5] P. E. Green, "Protocol conversion," *IEEE Trans. on Communications*, pp. 257–268, Mar. 1986.

[6] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello, "Practical applications of an efficient time separation of events algorithm," in *Proc. ICCAD*, pp. 146–151, 1993.

[7] K. Khordoc, M. Dufresne, E. Cerny, P. A. Babkine, and A. Silburt, "Integrating Behavior and Timing in executable specifications", in *Proc. CHDL*, pp. 385–402, 1993.

[8] L. Lavagno, "Synthesis and testing of bounded wire delay asynchronous circuits from signal transition graphs," Tech. Rep. UCB/ERL M92/140, U.C. Berkeley, Nov. 1992.

[9] A. J. Martin, "Programming in VLSI: From communicating processes to delay-insensitive circuits," in *UT Year of Programming Institute on Concurrent Programming* (C. A. H. Hoare, ed.), pp. 1–64, Addison-Wesley, 1990.

[10] K. L. McMillan and D. L. Dill, "Algorithms for interface timing verification," in *Proc. ICCD*, pp. 48–51, 1992.

[11] C. J. Myers and T. H.-Y. Meng, “Synthesis of timed asynchronous circuits,” *IEEE Trans. on VLSI Systems*, vol. 1, no. 2, pp. 106–119, June 1993.

[12] J. A. Nestor and D. E. Thomas, “Behavioral synthesis with interfaces,” in *Proc. ICCAD*, pp. 112–115, 1986.

[13] C. D. Nielsen and A. J. Martin, “Design of a delay-insensitive multiply-accumulate unit,” *INTEGRATION, the VLSI journal*, no. 15, pp. 291–311, 1993.

[14] K. Okumura, “A formal protocol conversion method,” in *Proc. ACM SIGCOMM*, pp. 30–37, 1986.

[15] W. Reisig, *Petri nets: An Introduction*, Springer-Verlag, Berlin, 1985.

[16] L. Y. Rosenblum and A. V. Yakovlev, “Signal graphs: From self-timed to timed ones,” in *Proc. of the Intl. Workshop on Timed Petri Nets*, pp. 199–207, July 1985.

[17] P. Vanbekbergen, *Synthesis of Asynchronous Controllers from Graph-theoretic Specifications*. PhD dissertation, Katholieke Universiteit Leuven, Sept. 1993.

[18] A. V. Yakovlev, “On limitations and extensions of STG model for designing asynchronous control circuits,” in *Proc. ICCD*, pp. 396–400, 1992.