Assessing the Feasibility of Interface Designs before their Implementation

Marco A. Escalante Nikitas J. Dimopoulos
Department of Electrical and Computer Engineering
University of Victoria
PO Box 3055, Victoria BC, CANADA V8W 3P6
e-mail: marco@sirius.uvic.ca

Abstract — During the design of microprocessor-based systems, once the system architecture has been decided and the major components (processors, memories, IO devices) have been selected from a component library, it is necessary to design interface logic to integrate the system. Such an interface design can be carried out based on the protocols used by the components. This paper addresses the problem of determining the feasibility of a design prior to synthesis. A design is called *feasible* if it achieves the desired functionality and satisfies the given environmental constraints. Because timing is an important aspect of a correct design, protocols are described using timed signal transition graphs, an interpreted Petri net. It is shown here that the feasibility of designs whose corresponding behavior is periodic can be studied using a technique called timing analysis for synthesis.

I. Introduction

As the complexity of hardware systems increases, techniques that facilitate their design and verification are invaluable to hardware designers. The DAME project [3] aims to automate the design of microprocessor-based systems. 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 library according to system-level design constraints such as type of application, throughput, cost, etc. The next step is system integration, during which DAME designs the necessary glue logic to interconnect the major components that comprise the system. In this paper we address the problem of verifying that such an interface design is *feasible* before synthesis is attempted, i.e. that the interface generates the necessary events at the expected times to accomplish the intended inter-component communication. Thus it is possible to avoid the design-synthesis-verification cycle: a design is first synthesized, then checked against its specification, and if verification fails the process is repeated.

In order to be able to describe the protocols used by off-the-shelf microprocessor components we have developed a representation of timed behaviors based on a Petri net model which allows us to reason about circuit *delays* and environmental *timing constraints*.

Our model is suitable for symbolic timing analysis that finds the tightest bounds on the unknown interface path delays before the actual circuit is implemented [4]. Delay-insensitivity [1] is a special case of circuit design in which timing constraints are satisfied by any implementation regardless of the circuit delays. 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 satisfy timing constraints for proper operation.

In section II we survey related work. Our timed representation and the symbolic timing analysis is presented in section III. The formulation of the interface design as the *merging* of protocol graphs is discussed in section IV. Future directions are offered in section V.

II. Related work

Signal transition graphs or STG's, a Petri net based representation formalism, have been used to describe the behavior of asynchronous control circuits [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 [9] identified the need of dealing with timing constraints in the design of interfaces. Recently work [12, 13] has been done in extending STG's to model circuit delays. Orbital nets [12] are based on discrete time and thus cannot handle dense time. Vanbekbergen's timed STG's [13] use real compact intervals to describe timing information, but the algorithm to compute time event separation does not always find the tightest bound.

However none of the aforementioned approaches can deal directly with unknown delays, thus they are unsuitable to study properties of designs before synthesis. In the following section we present our model which overcomes this problem.

III. Timed representation of protocols

Microprocessor components transfer information in the form of signals through wires that interconnect their ports. 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.

A. An example

Fig. 1 shows a read interface between a CPU and a RAM. Names of input ports are underlined. Signal transition graphs with time labels on the precedence links are used to represent the read protocols of both components (see Fig. 1b). A piece of data is transferred from the RAM output *dat* port to the CPU input <u>*dat*</u> port. Because in general it is not possible to observe transitions on data signals, control signals are required to annunciate transitions on the data lines. Data lines switch from a valid state to invalid (denoted by ↑) and viceversa (↓), and control lines are asserted (+) or negated (–).

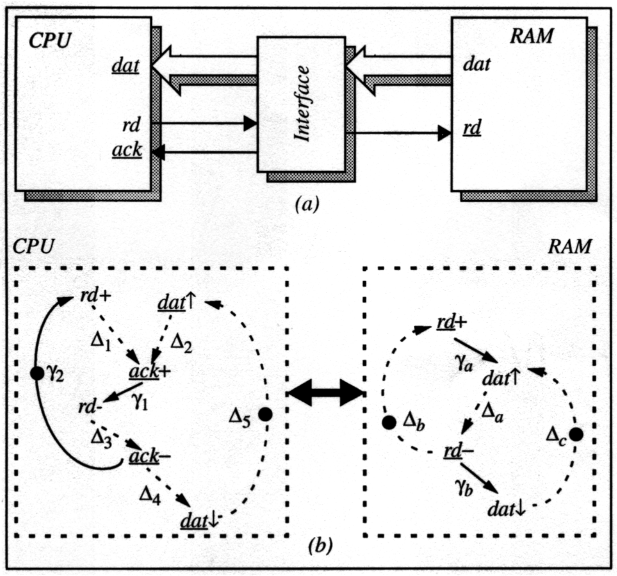

Fig. 1. Memory read interface: (a) structural view; (b) behavioral view.

Two types of precedence links are used to describe the partial ordering of actions in the protocols: operational links (solid lines) describe the component circuit delays; constraint links (dotted lines) specify the expected behavior of the environment for proper operation. Links are labelled with real compact intervals $[\tau_{min}, \tau_{max}]$. Constraint and operational labels are denoted by Δ_i and γ_i respectively. For example, the RAM read protocol describes the following two operations: after a positive transition is observed at the input port $\underline{rd}$ (a data request action), a piece of data will be put in port *dat* after a delay $c_a \in \gamma_a$ (an operational link); and after *dat* becomes valid, $\underline{rd}$ should remain asserted for any $d_a \in \Delta_a$ for proper operation (a constraint requirement). The CPU protocol controls the transfer using a pair of read/ack signals, while the RAM only defines a read control signal which is expected to remain asserted for a certain minimum duration (access time). Protocol conversion is required in the interface design.

The interface must provide a suitable environment that conforms to the specification by generating the input transitions of both protocol graphs. The design of the interface can be viewed as adding appropriate operational links so that the constraints are satisfied and the purpose of the protocol is accomplished (called semantic seed in [10]). In this example, the purpose of the read protocol is to transfer data from the RAM to the CPU.

In the following subsection we formalize 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.

B. *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 compact interval $\lambda \in I$. (N is the set of the naturals and I is the set of 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 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 to account for 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 visible 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, a constraint violation occurs.

C. *Ports, signals and signal transitions*

Ports are designated by unique names. Input port names are underlined (e.g., $\underline{a}$), while output port names are not. Signals carry the values of ports through wires. Let X be the set of input signals and Z the set of output signals of a circuit. The set of signals is $Y = X \cup Z$.

Y is partitioned into the set of *control* and *status* signals Y_c and Y_s. While every event on a control signal is manifested as a signal transition, an event on a status signal might not be accompanied by a signal transition (e.g. when the value of a data signal is the same in successive transactions). Thus status signals are not observable in general.

Transitions of control signals can be from negated to asserted (+) and from asserted to negated (–), while transitions of status signals can be from invalid to valid (↑) and viceversa (↓). The set of signal transitions[1] is $A = Y_c \times \{+, -\} \cup Y_s \times \{\uparrow, \downarrow\}$. A signal transition ($\underline{a}$, +) is written in short $\underline{a}$+. An arbitrary transition on port a is written as $a!$, and the complementary transition of $a!$ is written as $a!^*$.

D. *Timed signal transition graphs*

STG's are Petri nets whose transitions are interpreted as signal transitions. A timed STG is a triplet $\langle TPN, Y, \Delta \rangle$ where TPN is a timed Petri net, Y is as set of signals, and $\Delta: T \rightarrow A \cup \{\varepsilon\}$ is a labelling function which assigns a signal transition $a \in A$ or the anonymous transition ε to each transition $t \in T$ of the net.

Not every interpretation of a Petri net describes a correct behavior of a circuit (e.g., if two successive transitions of the

1.For the sake of clarity we have simplified our more elaborate signal representation scheme [5].

Petri net are labelled with the same signal transition). The *validity* of an STG is checked by ensuring that its corresponding state graph is consistent [13]. The validity of timed STG's is further discussed in section IV–A.

Definition 3.1.- A timed STG is said to be time-consistent if no constraint place flags a violation during any possible execution of the STG.

E. 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.

Interval arithmetic is used to write constraint equations. 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:

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

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

In the sequel we consider the subclass of marked graphs. In a marked graph, every place has a single input transition and a single output transition. Thus places can be drawn as links between two transitions. Consider transition d in Fig. 2 with 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 Fig. 2. Then 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 visible to d. This occurs within the interval $max\,(\tau_a + \gamma_1, \tau_b + \gamma_2, \tau_c + \gamma_3)$.

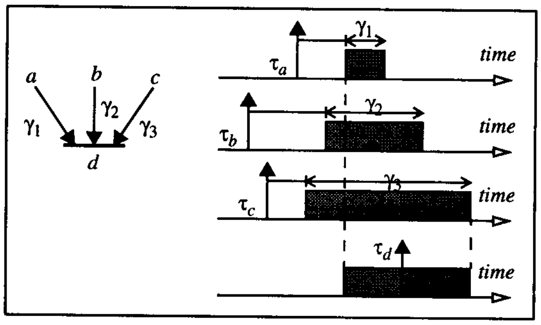

Fig. 2. Firing of a transition.

A constraint place between two transitions a and b (see Fig. 3) 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, it suffices to know bounds on the time separation from $\tau_a{}^i$ to $\tau_b{}^i$, written as $\tau_b{}^i - \tau_a{}^i$.

Definition 3.1.- A constraint place is time-consistent if for all occurrences i:

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

An STG is time-consistent *iff* all its constraint places are time-consistent. To compute the time interval difference in Eq. 1, we unfold the cyclic STG starting from the initial marking. The resulting unfolded graph is acyclic and infinite.

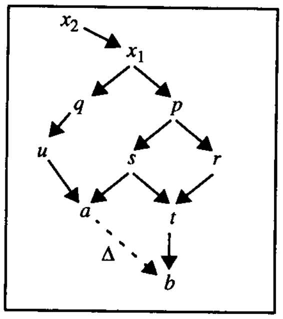

Fig. 3. Fork transition for constraint Δ.

Fig. 4 shows a simple protocol between two signals and its corresponding unfolded graph. In our application the execution of a protocol graph results in periodic behavior. Thus 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.*

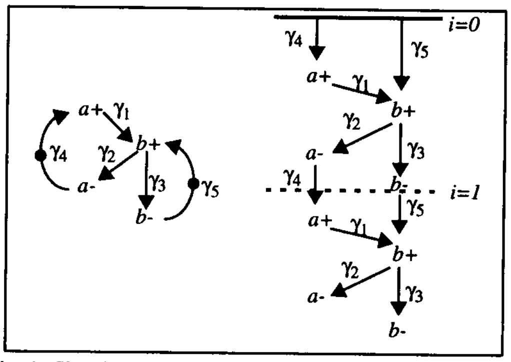

Fig. 4. Signal transition graph and its unfolded acyclic graph.

Definition 3.2.- 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 Fig. 3 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 Fig. 3 is also a fork transition for Δ.

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 $a{+}^i$ in Fig. 4 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 $a{+}^i$ is $b{+}^{i-1}$.

Eq. 1 involves the subtraction of interval expressions, each possibly containing *max* terms. Thus Eq. 1 is a nonlinear interval expression. The constraint satisfaction problem can be solved by solving first a finite set of subproblems [8]. 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 Latin letters (e.g., $c \in \gamma$).

We now state the timing analysis for synthesis 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 construct linear subproblems 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 $d_j \in \delta_j$. Note that according to Eq. 1, $y_k \in \Delta_k$. Then 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 $\{(c_1, \dots c_M, d_1, \dots d_N)\}$ which, when non-empty, is delimited by a (possibly unbounded) convex polytope [11]. Let $poly = \{(c_1, \dots c_M, d_1, \dots d_N)\}$ be the union of all the polytopes generated by the particular solutions. The total solution is the largest set $\{(d_1^*, \dots d_N^*)\}$ such that $\{(c_1, \dots c_M, d_1^*, \dots d_N^*)\} \in poly$ for all values $c_i \in \gamma_i$.

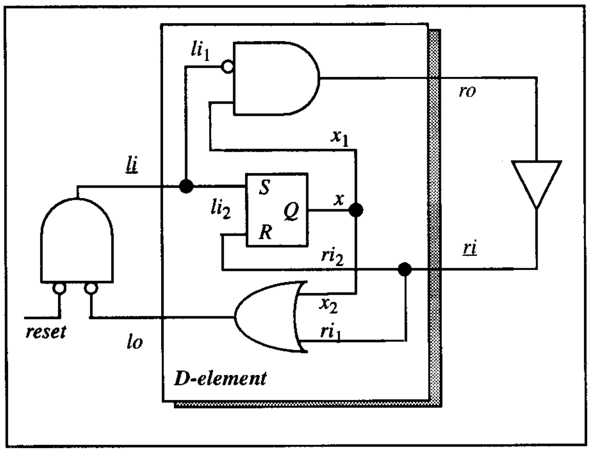

Fig. 5. Circuit implementation of the D-element.

F. Example

Consider the circuit implementation of a D-element shown in Fig. 5 which was reported in [6]. The D-element synchronizes two components that use handshakes to communicate. The left handshake $\underline{li}$+→lo+→$\underline{li}$–→lo– is interspersed with the right handshake ro+→$\underline{ri}$+→ro–→$\underline{ri}$–. State variable x is used to differentiate the two half cycles. Both the AND gate with inverted inputs and the buffer outside the D-element simulate the environment by generating the desired ack transitions after a gate delay.

Fig. 6 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 $\underline{li}$+ 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 $\underline{li}$–. Now the AND gate of the D-element causes transition *ro*+, which is propagated to $\underline{ri}$+. The flip-flop is reset, which subsequently produces the sequence *ro*–→$\underline{ri}$–→*lo*–. If a transition is propagated through different paths to different parts of the circuit, new transitions are created to take into account that the paths may have different delays. For example, transitions $\underline{li}_1$+ and $\underline{li}_2$+ represent the arrival of $\underline{li}$+ at the AND gate and flip-flop respectively (see also Fig. 5).

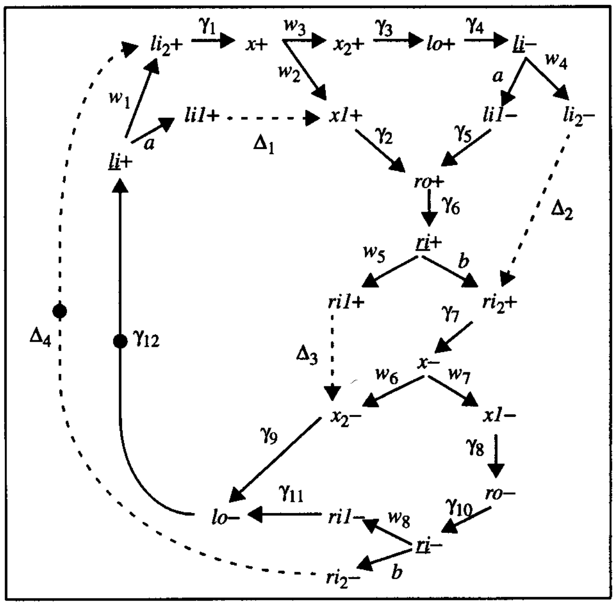

Fig. 6. Behavior of the D-element.

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 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 [7] suggested to assume isochronic forks, i.e. that the delays of forked transitions generated from a common transition that branches out into different paths are negligible compared to other delays; 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. Because Hulgaard's procedure can check the constraints only for known values of α and β, the intervals α and β that satisfy the constraints are found by trial and error. It is not clear that in general all values for the unknown delays can be found using this procedure.

Our symbolic timing analysis on the other hand finds all values for α and β that satisfy the constraints directly. First we write the four constraint equations corresponding to each Δ_i. For example the equation for constraint Δ_2 (with *x+* being the fork transition) is written as follows:

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

Note that for the given interval ranges, $\omega_2 + \gamma_2 < \omega_3 + \gamma_3 + \gamma_4 + \alpha + \gamma_5$ is always satisfied. 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 our timing analysis proves (see [5]) that all the constraints are satisfied if $\alpha = [0, 2]$ and $\beta = [0, 4]$. Therefore the circuit will function properly even if the isochronic fork assumption is violated.

IV. Timed asynchronous interface design

The interface design conceptualization is facilitated by a timed framework such as the one discussed in section III. 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, 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 the 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 also must be satisfied.

A. Valid specification

A valid specification describes a correct behavior considering both the circuit and its environment.

Definition 4.1.- Let $S = \langle PN, Y, \Delta\rangle$ be a timed STG. S is said to be a valid specification if it 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.

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 marked graph.

B. Interface design and STG feasibility

A correct interface implements the expected environment in both protocol graphs by generating the necessary input transitions. 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 status lines can be used only in conjunction with control transitions to generate new control events (remember that status transitions are not observable in general).

A semantic specification is a valid STG that describes constraints on selected signal transitions of the specifications.

Definition 4.1.- Let TS_1 and TS_2 be two valid 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 [10]. For example, Fig. 7a shows the semantic specification for a data transfer. In words, it describes that in a data transfer cycle (in this case a read cycle) it is expected that a piece of data will be transferred from source to destination.

Definition 4.2.- Given two valid specifications of two protocols together with the 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. Interface operational links do not sink to output transitions of the protocol specifications.
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 part of the interface design. Condition 2 forbids adding certain operational links as mentioned above. 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 considered feasible.

Definition 4.3.- A complete STG is called feasible if it is time-consistent.

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

It is possible that several interface designs for a given system are feasible. In the following example we show how different interface designs can be measured by comparing the solutions of the corresponding timing analysis for synthesis.

C. Read interface design

A design representing the read interface of Fig. 1 is shown in Fig. 7. The semantic specification (Fig. 7a) specifies that data will be transferred from source to destination. The *complete* STG representing the interface design is shown in Fig. 7b. The read protocols used by the CPU and RAM devices are subgraphs of the *complete* graph (cf. Fig. 1b). New thick lines with δ labels describe the interface path. The added interface links are compliant with condition 2 of Definition 4.2.

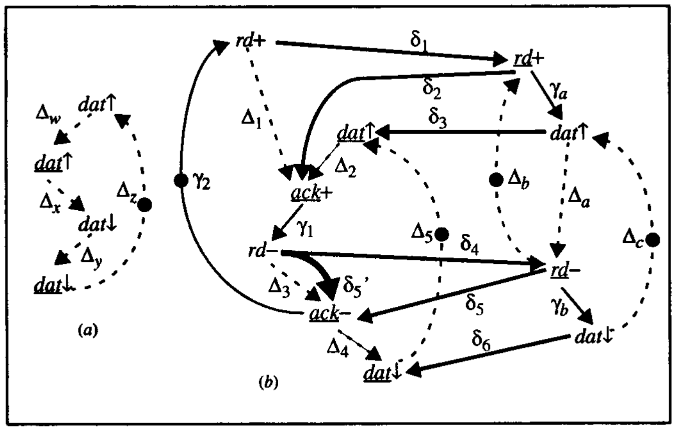

Fig. 7. Read memory controller: (a) semantic specification; (b) interface design.

To check if the interface is feasible we apply the timing analysis for synthesis procedure. There is a fork transition for every Δ constraint. For instance, the constraint equation for Δ_c is $\delta_5 + \gamma_2 + \delta_1 + \gamma_a - \gamma_b \subseteq \Delta_c$ where $\underline{rd}-$ is the fork transition.

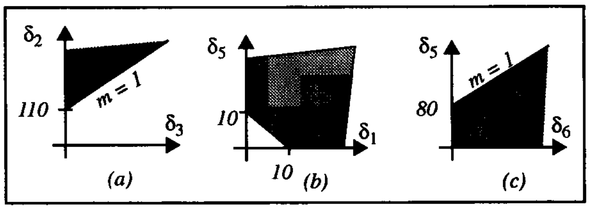

Fig. 8. Solution polytope projections.

Selected projections of the non-empty solution polytope corresponding to γ_1 = [10, 20], γ_2 = [10, 200], $\gamma_a = \gamma_b$ = [0, 100], set-up and hold constraints $\Delta_2 = \Delta_3$ = [10, ∞), access time Δ_a = [100, ∞), and other Δ_i's being [0, ∞) are shown in Fig. 8. The interface is clearly not delay-insensitive, otherwise all projections would cover the positive quadrant. Fig. 8a also shows that $\delta_2 - \delta_3 \geq 110$, i.e. δ_2 and δ_3 are not independent of each other. Consider a slightly different scenario: instead using δ_5 to generate transition $\underline{ack}-$ from $\underline{rd}-$, let us use transition *rd–* (delay δ_5' in Fig. 7). This new design is also feasible. Selected projections are shown in Fig. 8. Note that in this case δ_5' must be at least 10ns. The former design can accommodate more variations on the interface delays and it should be preferred over the second design.

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

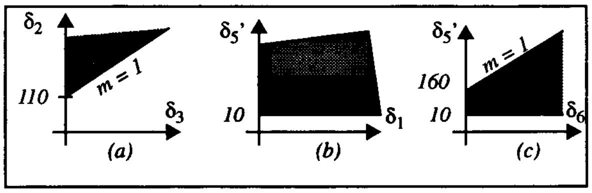

Fig. 9. Solution polytope projections for the second scenario.

V. Conclusions

DAME, a microprocessor-based-systems designer tool, represents components at a finer detail, the component protocol, so that during system integration it can design the required interface circuitry. In this paper we presented 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, achieves its purpose (described by a semantic specification), and generates a correct environment for the components to be connected (described by timing constraints specified in the protocols). Semantic and protocol specifications are represented uniformly in DAME's framework as timed signal transition graphs. 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. Finally the solution of the timing analysis for synthesis procedure can also be used to compare several designs that implement a given interface. Currently we are investigating knowledge-based techniques to efficiently find feasible designs given the component protocols.

Acknowledgements

This work was partially supported by NSERC operating and strategic grants. The first author was also supported by a University of Victoria Fellowship.

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] M. A. Escalante, "Bus arbitration modelling and design in DAME," M. A. Sc. thesis, University of Victoria, 1991.

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

[5] M. A. Escalante, "Assessing the feasibility of hardware designs", Technical Report ECE-95-01. University of Victoria, 1995.

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

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

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

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

[11] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[12] T. G. Rokicki, *Representing and Modeling Digital Circuits*. PhD dissertation, Stanford University, Dec. 1993.

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