Timed Asynchronous Interface Design in Microprocessor-based Systems

M. A. Escalante and N. J. Dimopoulos

Department of Electrical and Computer Engineering
University of Victoria

Abstract

Asynchronous techniques have been rediscovered as an attractive alternative to digital design due to the development of new formalisms such as signal transition graphs, CSP programs, and event labelled structures. Until recently the main research thrust was expended in the area of delay-insensitive circuits for which only sequence and not timing is sufficient to describe a system's behaviour. In this paper the design of interfaces in microprocessor-based systems is explored in the context of a timed asynchronous design. An extension of signal transition graphs is used not only to specify the timed behaviour of the interface protocols found in microprocessor chips but also to design the interface control path. A methodology is proposed to deal with the time constraints that crop up in the protocols of ordinary microprocessor families: environmental constraints are transformed into constraints on the interface delays which can be found ahead of the implementation stage. A physical implementation of the interface is correct if it satisfies such set of interface path delay constraints.

1 Introduction

The design of microprocessor-based systems differs from the VLSI approach in that off-the-shelf microprocessor components are used as the basic building blocks instead of silicon. The final objective is not the synthesis of logic functions that implement the system but the design of interfaces to glue the components that comprise the system. This is called the *interface design problem.*

Design automation is desirable not only to eliminate errors during the clerical tasks of the design process but also to keep up with the technological drive that is producing new chips at an unprecedented rate. One way to bridle the complexity is to decompose the design problem into several phases which only require to look at part of the information at a time. A top-down approach to design in which the final system is obtained as a sequence of successive refinements from the initial system specifications is attractive mainly because it allows the designer to concentrate on the important issues at some phase of the design process while hiding the unnecessary details.

In this paper we propose a methodology to break down the interface design problem into two simpler subproblems, an interface design phase in which the logical design of the interface is carried out and an interface implementation phase in which a physical design is produced. The main contribution of this paper is a procedure that, during the design phase, computes a set of constraints on the interface path delays using the environmental time constraints specified by the interface behaviour of the microprocessor components. In this manner, such a set of constraints can be used to detect design problems in advance of the implementation (i.e. negative path delays), to guide the implementation phase (i.e. time-driven partitioning, placement, and routing), and to verify that the implementation timed behaviour of the final interface circuit satisfies the design constraints.

In section 2, related work is surveyed in two directions: the automation of the design of microprocessor-based systems, and the new developments in asynchronous digital design. Basic concepts are introduced in section 3. In section 4, the interface design problem is briefly discussed. Section 5 outlines a procedure to cope with time constraints: environmental constraints are transformed into constraints on the interface path delays. This paper concludes with the final remarks and future work.

2 Related work

A microprocessor-based system can be viewed as a collection of components which operate independently of one another, sometimes needing to synchronize and communicate with the rest of the system through communication structures called buses. The interface design problem arises during the system integration design phase when components need be amalgamated into one entity. The design of such an interface often involves not only matching the physical and electrical levels of the signals but also converting protocols.

MICON developed at CMU [1] is an expert system that designs single-board computer systems from system level specifications such as type of application, cost, throughput, etc. MICON uses the system specifications to select components from its component database. A design cycle is defined such that the CPU, memory and I/O devices are chosen in such order.

MICON solves the interface design problem by storing in its component database complete subsystems called *templates* which contain not only a basic component (i.e. CPU, memory, I/O device) but also the additional glue logic to interface it to predefined communication structures called generically Micon buses. A Micon bus follows closely the standard bus used by a microprocessor family of components. Once the CPU has been selected, the other component templates are chosen from the group of templates sharing the same Micon bus. Then for, say, a memory chip there are as many templates as Micon buses. The MICON

design process is then reduced to choosing the appropriate subsystem aggregates and connecting them together. Finally MICON outputs its design as a netlist to other VLSI tools that carry out the physical design.

Although in MICON the interconnection of component templates is straightforward, the updating of the component database presents a problem because as the number of devices n grows, the number of templates that need be stored in the general case[1] is $O(n)$. A simplification in the design of single-CPU systems is to limit the Micon buses to the number of CPU buses. The number of templates for memory and I/O devices is then reduced to $O(m)$, where m is the number of CPU's in the system [1].

In DAME [5] we suggest that a finer grain modelling of components can prove helpful in reducing the uncontrolled growth of the component database: The components' interface behaviour is represented by interface protocols. Thus protocol templates are used instead of component templates. Because there are but a limited number of protocols, fewer general design rules are needed to cope with the interface design problem. In addition, systems with more complex architectures than single-CPU boards can be designed.

An interpreted Petri net called signal transition graph (STG) has been proposed in the literature to represent behaviour of asynchronous digital circuits [4, 12]. Nodes in the graph correspond to signal transitions and links between nodes describe a precedence relation between transitions. STG's were first applied to delay-insensitive designs, in which unbounded positive finite delays are associated with the links. Several synthesis procedures have been developed that solve the synthesis problem of circuits described by STG's. The traditional problems of asynchronous circuits, namely races and hazards, can be analyzed and dealt with using the new techniques [6]. Recently STG's have been extended to describe timed (i.e. delay-sensitive) asynchronous circuits [9].

Other work has been done in the specification and design of the interface control circuitry [10, 2, 3]. In this paper the effect of time constraints in the interface design in microprocessor-based systems is considered. The time constraints specified by the interface's environment are converted into constraints on the interface path delays. The interface design problem is posed as the logical design of the interface, represented as a timed STG describing constraints on the interface path delays, followed by its physical implementation. Any correct interface implementation must satisfy the set of constraints on the interface path delays.

3 Protocol specification

Microprocessor components transfer information in the form of signals through wires that connect their ports. Input ports accept incoming signals generated in output ports. A protocol enforces the correct transfer of information by defining the order and timing of elementary operations called actions. Signal transitions are used to encode the actions. We use an abstraction of STG's to express the behaviour of component interface protocols called action graphs [5]. After the interface design is produced as the combination of the action graphs of the components to be interconnected, the *merged* action graph is transformed into a timed STG.

3.1 Ports, signals and signal transitions

Ports are designated by unique names. Input port names are written as $\underline{a}, \underline{b}, \underline{c}$ while output port names are written as a, b, c. Signals carry the values of ports through wires. Let X be a set of m input ports and Z the set of n output ports of a circuit. The set of signals is $Y = X \cup Z$.

The alphabet $A = Y \times \{+,-\}$ is the set of all (binary) signal transitions[2]. A signal transition $\{\underline{a}, +\}$, written $\underline{a}+$, indicates a positive transition of the signal value at input port $\underline{a}$. Signal transitions $a+$ and $a-$ are called *opposite* transitions. When the type of transition is not important we use the notation $a!$ to stand for $a+$ or $a-$.

Delays are modelled by using different signal transitions at the ports connected via the module in question. A module may be a wire, a buffer, or logic. For example a wire delay is shown in Figure 1. The two signal transition frames are the initiation of an output signal transition in the sender and the reception of the corresponding input signal transition at the receiver. The only assumption made is that signal transitions occur instantaneously although they may take a finite time to propagate through wires.

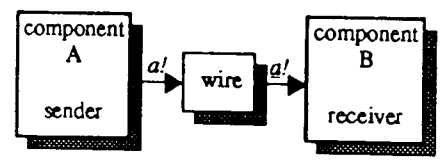

Figure 1. Wire delay model.

3.2 Signal transition graphs

STG's are Petri nets whose transitions are interpreted as signal transitions. A marked Petri net [4] is a quadruple $PN = \langle Tr, Pl, F, M_0 \rangle$ where Tr is a non-empty set of transitions, Pl is a non-empty set of places, $F \subseteq (Tr \times Pl) \cup (Pl \times Tr)$ is the flow relation between transitions and places, $M : Pl \rightarrow N$ is the marking function (N is the set of natural numbers) which assigns tokens to places, and M_0 is the initial marking. A transition $tr \in Tr$ is enabled by a marking M if all its predecessor places have at least one token. Every enabled transition may fire. After a transition fires, a new marking M' is obtained from M by removing one token from each predecessor place of tr and adding one token to each successor place of tr.

An STG [6] is a triple $\langle PN, Y, \Delta \rangle$ where PN is a marked Petri net, Y is a set of signals, and $\Delta : Tr \rightarrow A$ is a labelling function which associates each $tr \in Tr$ of the Petri net with an $a! \in A$.

A binary relation P (for precedence) on the set of signal transitions A can be defined as follows:

$a!\ P\ b!$ *iff* $a!$ immediately precedes $b!$,

i.e., $a!\ P\ b! \Rightarrow [\neg\exists c! \in A : a!\ P\ c! \wedge c!\ P\ b!]$

The pair $\langle A, P \rangle$ is a digraph where A is the set of nodes, P is the set of links, and there is a link from $a!$ to $b!$ *iff* $a!\ P\ b!$. Nodes and links in the digraph can be

1. Each component has a different interconnection.

2. The transition set can be naturally extended to describe multi-valued signals [13]. For example a signal could be in one of three possible states: asserted, negated, and tri-stated.

associated with transitions and places of a Petri net respectively. The resulting Petri net is a marked graph, which generates the class of conjuctive STG's [4]. Thus a conjuctive STG can be represented as the triple $\langle A, P, M_0 \rangle$ where A is the set of signal transitions, P is the precedence relation, and $M_0 \subseteq P^{\infty}$ is the initial marking. Concurrency can be expressed with conjuctive STG's but not choice. In this paper we only consider the simplified conjunctive model of STG's.

Although the design of a delay-insensitive microprocessor has been reported in [8], most microprocessor components nowadays use delay-sensitive protocols. To express time, a label $t = [t_{min}, t_{max}]$ is associated with each link[3] of the STG that assigns a minimum and a maximum time value to the occurrence of $b!$ after $a!$ whenever $a!$ P $b!$. Thus, in the underlying Petri net of a *conjuctive* timed STG, when a transition fires the token assigned to a successor place remains there only during interval t after the transition.

In a pure causal link, the associated time label is $[0,\infty)$. Frequently labels are under-specified, in which case the unspecified minimum/maximum values are assumed to be zero/infinite. The label of a link with only a minimum specified time is $[t_{min}, \infty)$. Likewise a link with only a maximum specified time has an associated label $[0, t_{max}]$. The values of both t_{min} and t_{max} are non-negative (real) numbers.

3.3 Path length

Because labels represent time intervals, interval arithmetic facilitates the label manipulation needed in the constraint satisfaction procedure discussed in section 5.3. Let I be the set of real compact intervals. In general an interval operation $\otimes$ is defined by [11]:

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

for $\alpha, \beta \in I$ and $a, b \in R$.

There exist exact values a, b in the interval of the operation, but we are interested in finding the smallest interval $\alpha \otimes \beta$ which contains all possible $a \otimes b$. In particular the calculation of interval addition, subtraction, minimum and maximum are given by:

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

where interval α is $[\alpha_{min}, \alpha_{max}]$. The minimum and maximum values of interval α are generically denoted α_m. Observe that subtraction is not the inverse operation of addition, as it is the case in *real* arithmetic.

The *length* of the path p from transition $a!$ to transition $b!$, written Σp, in a conjunctive STG is computed recursively as follows:

- $\Sigma p = [0,0]$ if p is the empty path (i.e., $a! = b!$)
- $\Sigma p = t$ if there exists only one link between $a!$ and $b!$ whose label is t.
- $\Sigma p = max\,(\Sigma p_1, \Sigma p_2, ...)$ if $p = p_1 \mid p_2 \mid ...$ where p_i are all the paths that share the same fork and join transitions $a!$ and $b!$.
- $\Sigma p = \Sigma p_1 + ... + \Sigma p_n$ if $p = p_1 \cdot ... \cdot p_n$ such that there are no parallel paths between the head of p_i and the tail of p_j for all i, j through the path.

The *max* terms in the length of a path composed of parallel subpaths arise because a join transition is enabled until all of its preceding transitions have occurred. In a disjunctive STG, where the join transition is enabled when one of its preceding transitions has occurred, the max terms in the computation of Σp are replaced by min terms (cf. [7]).

Paths with conflicting time labels indicate inconsistencies in the specifications. For example in Figure 2 the interval relation $(t_1 + t_2) \cap (t_3 + t_4) \neq \phi$ must be satisfied, otherwise both paths would have incompatible time duration, i.e. the duration from $a!$ to $d!$ through $b!$ would be outside the range of the path through $c!$ and since tokens remain alive only for the duration indicated by the path length, $d!$ could never be executed.

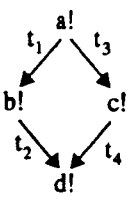

Figure 2. Checking conflicting paths.

4 Interface design

In this section we summarize our approach to the interface design. The interested reader is referred to [5] for more details.

4.1 Interface design problem

Given a system specification which includes type of application, throughput, cost, etc., the designer must decide the system's architecture, choose the appropriate components and come up with the necessary glue logic to incorporate the components into the system. The interface design problem arises when several components are to be interconnected together to build up a system.

In the case of microprocessor-based systems, one can decompose the general component interconnection problem into the individual connections between a component and a bus (see Figure 3). A bus can be the CPU bus, a standard system bus (i.e. VMEbus, Futurebus, etc.) or any of a hierarchy of buses that may appear in the system. For example, incorporating memory to a single-CPU system can be viewed as the interconnection between the memory chip and the CPU bus. Observe that a CPU is interfaced trivially to its own CPU bus.

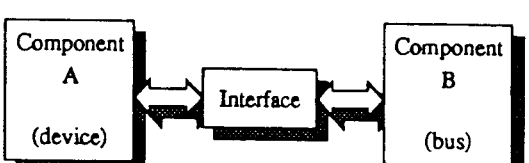

Figure 3. Interface design problem.

Protocol descriptions of hardware components are often offered by the manufacturer in the form of timing

3. Although for conjuctive STG's each time label can be thought of being associated with a place of the marked graph, in general a label is assigned to each link from a place to a transition or from a transition to a place of the underlying Petri net (cf. [14]).

diagrams [2, 3]. Figure 4a shows the timing diagram of a fully interlocked handshake bus arbitration protocol. A potential master requests the bus from the arbiter by asserting its REQ signal. When the input signal ACK is asserted by the arbiter, the requestor can take over the data transfer bus DTB. At the completion of the transfer the master gives up the bus by negating REQ and waits until the arbiter has negated ACK to initiate another cycle. Figure 4b shows the action graph corresponding to the timing diagram. Nodes in the graph represent actions of the protocol and links describe the precedence between actions. The token indicates the initial state of the protocol. Observe that it is not clear from the timing diagram if *r+* must wait for transition *a-*. The graph formalizes the ambiguous behaviour of the protocol by ascertaining that *r+* occurs only after *a-*.

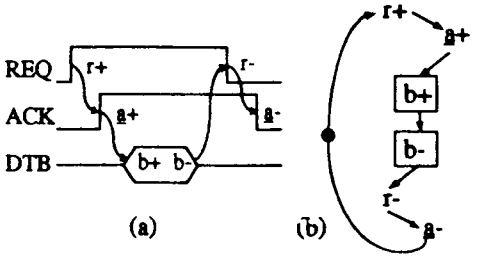

Figure 4. Fully interlocked handshake bus arbitration protocol: (a) timing diagram; (b) STG.

4.2 Merging signal transition graphs

Consider the interface between two devices shown in Figure 5. Each device has one input signal *i* and one output signal *o*. The structural description (in Figure 5a) shows that the output port in one component is connected to the input port of the other component. Connector or *C* blocks are intended to leave the conditioning of the electrical and logical characteristics of the signals to a subsequent design phase.

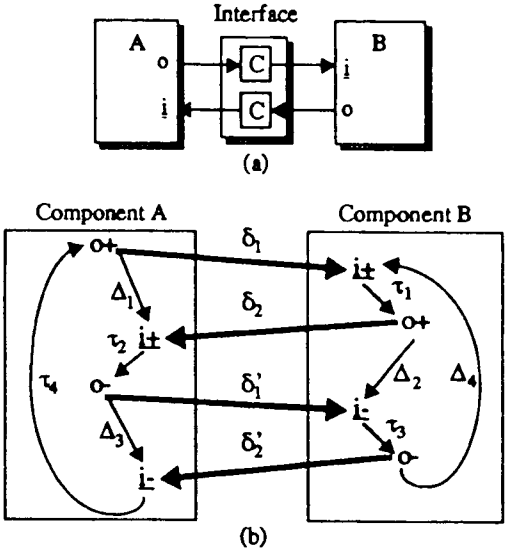

Figure 5. Ideal interface: (a) structural description; (b) behavioral description.

The behavioral description of the interface describes two handshake protocols. The thick links represent the paths through the *C* blocks and wires. There is a delay associated with each interfacing link. In this example, a pair of delays δ_i, δ'_i is associated with each interface path through the *C* blocks. In general, the values of a pair of delays may be different. This is the case when a link represents combinational logic which may respond with different speed to positive and negative transitions.

Although a classification of link labels is deferred until section 5.1, observe that τ and δ labels in Figure 5b correspond to propagation delays through physical circuitry, while Δ labels indicate required times that must be satisfied for correct operation. The purpose of the interface links is to provide physical paths from output to input transitions (i.e. to generate the input signals) that preserve the transition sequence in the original protocols, while satisfying the constraints.

5 Time Constraints

In this section we suggest a methodology to deal with time constraints using interval arithmetic. A procedure is proposed that determines a set of inequalities on the interface path delays which parameterizes the timed control behaviour of the interface. First we present a classification of labels.

5.1 Timing label classification

Time constraints play an important role in the interface design phase due to the restrictions they impose on the control sequence of the protocol. We distinguish between constraints imposed by the interface environment which are specified by the protocols of the devices to be interconnected, and constraints required by the interface basic building blocks such as the set-up and hold time of a latch used within the interface. Only environmental constraints are considered in this paper.

Links labels are classified according to the source of their minimum and maximum time values into the following groups:

- τ labels which represent known operational times in the component protocols (i.e. the time an output port takes to change in response to an input transition).
- δ labels which represent interface path delays
- Δ labels which represent environmental constraints on the protocols for proper operation, such as set-up and hold minimum times to avoid metastability.
- ε labels are specialized Δ causal labels between opposite input transitions, which are discussed further in section 5.2.

Links are named after their time labels. Thus a link with a τ label is called a τ link. Because τ and δ links have a correspondence to physical paths within a component or the interface respectively, they are called *operational* links. The τ and δ labels are intervals due to variations in the device fabrication process, operation temperature, etc. On the contrary, Δ and ε links are virtual links in the sense that there is no silicon associated with them, and are called *constraint* links.

5.2 Constraint satisfaction

Because there are no physical paths associated with constraint links but they represent time restrictions that

must be met for proper operation, their head and tail transitions must also be connected by paths with τ and δ links which provide a physical path between the two transitions. Thus it is possible to check if the physical path satisfies the restrictions imposed by the constraint link. Paths consisting only of operational τ and δ links are called *implementation* paths.

Two situations may arise in satisfying a constraint link with implementation paths as depicted in Figure 6. If the head of the constraint link ($a!$, $\underline{b}!$) is an output transition (Figure 6a), there must be one implementation path from $a!$ to $\underline{b}!$, otherwise it would be impossible to ensure the time precedence implied by the constraint label. Similarly if the head $\underline{a}!$ of a constraint link ($\underline{a}!$, $\underline{b}!$) is an input transition, there must be an output fork transition from which two implementation paths can be drawn to $\underline{a}!$ and $\underline{b}!$ (Figure 6b). The former situation is a special case of the latter in which the head of the constraint link is the fork transition and the implementation path preceding the constraint link is the empty path. In both cases there are two implementation paths from the fork transition to the head and tail of the constraint link, called *ph* and *pt* paths respectively.

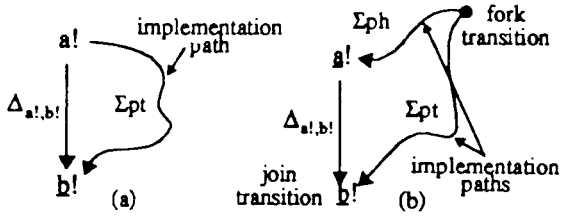

Figure 6. Implementing Δ links: (a) output-to-input Δ link; (b) input-to-input Δ link.

To satisfy the constraint link, the interval *difference* between the occurrence of the tail and the head transition of the constraint link must be within the interval specified by the constraint label. The occurrence of the head and the tail transition are controlled by the *ph* and *pt* paths. Thus the constraint satisfaction can be written as the interval expression $\Sigma pt - \Sigma ph \subseteq \Delta$, where a Δ constraint link is assumed for the sake of illustration without loss of generality. Two inequalities of the form $\Delta_{min} \leq f_a(\delta_{i,m}, \tau_{i,m})$ and $\Delta_{max} \geq f_b(\delta_{i,m}, \tau_{i,m})$ are produced from the interval expression, where functions f_i contain only sums and subtractions of linear and max terms on the minimum and maximum values $\delta_{i,m}$, $\tau_{i,m}$ of the labels of operational δ and τ links.

If the constraint link is causal ($\Delta = [0,\infty)$) and the path *ph* is empty, both inequalities are satisfied for finite operational delays. Therefore a causal output-to-input constraint link can be satisfied by any parallel implementation path. Moreover the Δ_{max} inequality is always satisfied for a causal constraint link even if path *ph* is not empty.

For example consider the interface between a CPU and a memory with only write operation shown in Figure 7. The interface consists of two *C* blocks as shown in Figure 7a. The STG describing the CPU protocol contains only output signals. Thus all its link labels are τ labels. Conversely the labels in the STG corresponding to the write-only memory chip are Δ and ε labels. One can identify Δ_2 and Δ_3 as the set up and hold times respectively of *dat* with respect to *wr*-, the transition that latches the data. Δ_1 is the width of the *wr* pulse. Usually only minimum values are given to these Δ labels. By making the appropriate interface connections (thick links in Figure 7b) the input transitions of the protocols are generated from the output transitions. The ordering of transitions in both protocols is preserved *iff* the operational times satisfy the constraint links.

The constraint link Δ_1 in Figure 7b is an input-to-input Δ link. Two implementation paths to the head and the tail of the constraint link are the path from *wr*+ to *wr*+, and the path from *wr*+ through *wr*- to *wr*-. Therefore $\tau_2 + \delta'_1 - \delta_1 \subseteq \Delta_1$ which can be expanded into two inequalities: $\delta'_{1,max} - \delta_{1,min} \subseteq \Delta_{1,max} - \tau_{2,max}$ and $\delta'_{1,min} - \delta_{1,max} \subseteq \Delta_{1,min} - \tau_{2,min}$. The interval expressions for the constraints Δ_2, Δ_3, ε_2 and ε_3 are $\tau_1 + \tau_2 + \delta'_1 - \delta_2 \subseteq \Delta_2$, $\tau_3 + \delta'_2 - \delta'_1 \subseteq \Delta_3$, $\tau_4 + \delta_2 - \delta'_2 \subseteq [0,\infty)$, and $\tau_3 + \tau_4 + \tau_1 + \delta_1 - \delta'_1 \subseteq [0,\infty)$ respectively.

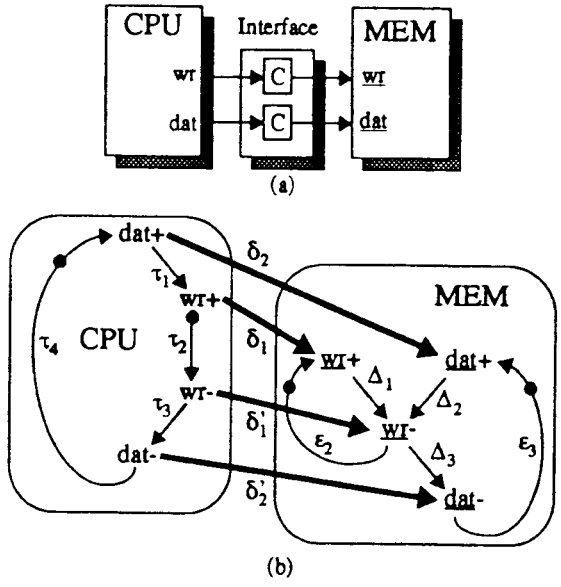

Figure 7. Simple memory-to-CPU interface: (a) structural description; (b) behavioral description.

5.3 Time constraint satisfaction procedure

In this section we propose a procedure that, given a merged graph representing the interface design, obtains a set of inequalities on the minimum and maximum values $\delta_{1,m}$ of the δ labels.

The minimum and maximum values of the operational labels are assumed to be known, and they must satisfy the constraint labels. However before the interface implementation is carried out, the δ labels are unknown. Therefore we rephrase the problem of time constraint satisfaction in the interface design to that of finding the δ intervals that satisfy the constraint links. In this manner it is possible to determine the interface's permissible delays in advance of its implementation. For instance, a negative maximum time in a δ label indicates a design problem which requires a redesign of the interface. Also suitable technologies can be selected to carry out the interface implementation based on the tightest required interface delays. The timing interface verifica-

tion task can also be facilitated by highlighting the critical delay paths.

Both Δ and τ labels are used to determine the allowable ranges on δ as delineated by the following:

Procedure: Given a conjuctive timed STG representing the interface design do:

1. For each Δ or ε link in the graph do until all output fork transitions are considered:

- Find an output fork transition from which two implementation paths *ph* and *pt* can be drawn to the head and tail of the constraint link.
- Write the interval expression $\Sigma pt - \Sigma ph \subseteq \Delta$ or ε.

2. Expand the interval expression into inequalities in which the variables are the minimum/maximum values $\delta_{i,m}$ of the δ labels.

3. Write for each δ label in the graph the inequalities $\delta_{i,m} \geq 0$ and $\delta_{i,max} \geq \delta_{i,min}$.

Let $\hat{\delta}$ be a vector containing the n unknown values of $\delta_{i,m}$. The result of applying the above procedure is a set of non-linear inequalities on δ due to the presence of the *max* terms. Every inequality containing M max terms, each holding m_i $(i = 1, \ldots, M)$ linear expressions e_k $(k = 1, \ldots, m_i)$ on δ, can be substituted by $\sum (m_i - 1)$ linear inequalities by selecting a winner w_i among the linear expressions for each max term and writing $w_i \geq e_k$ for $k = 1, \ldots, m_i \wedge k \neq i$. A winner choice for every max term defines one particular solution, whose corresponding expanded set of N linear inequalities can be written as the linear program [3]:

$$max\ f(\hat{\delta})$$
$$\text{Subject to } A\hat{\delta} \geq b$$

where $f(\hat{\delta})$ is the null function, with $A \in R^{N \times n}$ and $b \in R^N$. The solution of the linear program is the set of feasible points which, when non-empty, is bounded by a convex polytope. There are in total $\Pi\, m_i$ particular solutions. A solution vector δ^* must satisfy at least one of the particular solutions.

Unfortunately the complexity of reduction sketched above is not polynomial-time on the number of max terms in the system of $\delta_{i,m}$-inequalities. In [7] a similar problem was also shown to be NP-complete in which the interface timing is verified assuming that τ and δ are known subject to a set of constraints Δ.

6 Conclusions

In this paper a procedure that determines ahead of the physical implementation the time constraints on the internal delay paths of the interface was proposed. The procedure transforms the environmental constraints represented by Δ constraints and ε constraints into δ interface constraints. Such set of δ delay constraints can be used to: i) detect inconsistencies in the design before attempting to convert it into silicon (i.e. a negative delay is required in one path of the interface); ii) guide the lower synthesis stages (i.e. time-driven partitioning, placement, and routing); and iii) verify that the final implementation operates correctly.

With the procedure described in this paper it is possible to separate the integration design phase, that amalgamates the microprocessor components into a single system by designing interfaces, into a logical design subphase and an implementation subphase. A timed STG graph is the intermediate representation between the two subphases.

This work attempts to bridge the gap between the trends in design automation of microprocessor-based systems and the new developments in digital design techniques which are reaching a common framework to encompass both synchronous and asynchronous timing disciplines.

Acknowledgments

We are grateful to Dr. L. Lavagno for enlightening comments on a previous draft, and to Dr. A. Yakovlev for bringing [13] to our attention. Also we would like to thank the anonymous reviewers for their comments and corrections. This research has been supported in part by NSERC grants OGP-00041188 and STR-0134222. M. Escalante has also been supported by a University of Victoria fellowship.

References

[1] W. P. Birmingham and D. P. Siewiorek, "Single board computer synthesis," in *Expert Systems for Engineering Design*, chapter 5, pp. 113-139, Academic Press, 1988.

[2] G. Borriello and R. H. Katz, "Synthesis and optimization of interface transducer logic," in *Proc. ICCAD*, pp. 274-277, 1987.

[3] J. A. Brzozowski, T. Gahlinger, and F. Mavaddat, "Consistency and satisfiability of waveform timing specifications," *Networks*, vol. 21, pp. 91-107, Jan. 1991.

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

[5] M. A. Escalante, "Bus arbitration modelling and design in DAME: An expert microprocessor-based-systems designer," M. A. Sc. thesis, University of Victoria, 1991.

[6] L. Lavagno, "Synthesis and testing of bounded wire delay asynchronous circuits from signal transition graphs," Tech. Rep. UCB/ERL M92/140.

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

[8] A. J. Martin et. al., "The design of an asynchronous microprocessor," in *Proc. Decennial Caltech Conf. on VLSI*, pp. 351-373, 1989.

[9] C. Myers and T. H.-Y. Meng, "Synthesis of timed asynchronous circuits," in *Proc. ICCD*, pp. 279-284, 1992.

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

[11] H. Ratschek and J. Rokne, *Computer Methods for the Range of Functions*. Ellis Horwood, 1984.

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

[13] A. V. Yakovlev and A. I. Petrov, "Symbolic signal transition graphs and asynchronous circuit design," Tech. Rep. 395, University of Newcastle upon Tyne, Sept. 1992.

[14] J. J. Zhu and R. T. Denton, "Timed Petri nets and their application," in *Proc. MILCOM*, pp. 195-199, 1988.