Timing Analysis for Synthesis in Microprocessor Interface Design

Marco A. Escalante and Nikitas J. Dimopoulos

Department of Electrical and Computer Engineering
University of Victoria, BC CANADA
PO Box 3055, Victoria BC, V8W 3P6 CANADA

Abstract

Design automation techniques are playing an important role in controlling the complexity of system design. Our work is inscribed in the design automation of microprocessor-based systems which necessitates the design of interfaces for system integration. During the interface synthesis it is required to validate the timing of a design yet to be implemented. In this paper we present a novel methodology to timing analysis that can determine tight bounds on interface path delays based on the given timing information. The timing analysis for synthesis problem is formulated as a combinatorial optimization problem using interval arithmetic techniques.

1. Introduction

The design of interface circuits emphasizes the synthesis of control logic [4]. Other research work [9] indicates that controller design can benefit from a delay-insensitive design methodology which produces robust 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 [10]. This is particularly true in the design of microprocessor-based systems whose protocols specify deadlines to meet.

In this paper we discuss a Petri net based representation formalism that allows us to reason about known circuit path delays and environmental timing constraints. Although our approach shares similarities with other work in the area of interface synthesis and controller design, our aim is to break a recurrent problem encountered during synthesis: a solution must be first offered to be able to determine if it satisfies the design constraints. The main result of this paper is a novel approach to time analysis that breaks this cycle by finding bounds on the delays of the circuit to be synthesized *before* the actual circuit implementation takes place.

In the following section we survey related work. Our representation formalism is based on a timed Petri net which is presented in section 3. Section 4 is devoted to the discussion of the details of the timing analysis methodology. An example is given in section 5 to illustrate our approach. Finally future work is pointed 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.

MICON [3] is an expert system that designs single-board computer systems from system level specifications. MICON solves the interface design problem by providing in its component database complete subsystems called *templates*. A template contains not only a basic component but also the necessary glue logic to conform to a predefined MICON bus. Although in MICON the interconnection of component templates is straightforward, the price to pay is a potentially uncontrolled growth of the database due to the requirement of storing *N* different templates for each component where *N* is the number of MICON buses. Also as technology evolves it is likely that new MICON buses must be incorporated to the database.

In DAME [7] the components' interface behavior is used to design a suitable interface. Because there are but a limited number of protocols, fewer more general design rules can cope with the interface design.

Signal transition graphs or STG's, a Petri net based representation formalism, have been used to describe the behavior of asynchronous control circuits [6]. STG's were first applied to the design of delay-insensitive circuits which assumes unbounded wire and gate delays. Although

0-8186-5785-5/94 $03.00 © 1994 IEEE

a very robust design methodology, delay-insensitivity is not realistic to describe the behavior of microprocessor components.

Pioneering work by Nestor and Thomas [10] identified the necessity of dealing with timing constraints in the design of interfaces. Borriello [4] developed an interesting framework for interface design that is based on a formalized timing diagram language called WAVES, capable of representing different types of timing constraints. The synthesis procedure SUTURE initially constructs a design that is not necessarily correct and successively transforms it to meet the design constraints.

Recently some work has been done in extending STG's to incorporate the *bound* delay model, where actual gate delays are taken into consideration. Burns [5] developed a formalism called *event-rule system* to analyze the performance of asynchronous circuits. Myers and Meng [9] used a conservative estimate of gate delays and available environmental timing constraints to remove redundancies in the specification. 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, has been shown in [8] to be NP-complete. An algorithm that finds exact bounds on the maximum distance between two events is reported in [1] where the restriction that the graph be acyclic has been lifted.

In a related direction, STG's have been extended with timing constructs to support the description of both synchronous and asynchronous circuits in [12, 11, 7]. Their approach differ from other timed Petri net formalisms (cf. [2]) in that the time intervals are assigned to places instead of transitions, and in that different types of places are used to describe two similar but subtly different timing behaviors: environmental constraints and circuit delays.

In this paper we expand on previous work and present a novel approach to timing analysis for synthesis. We present a procedure capable of determining bounds on the path delays of the interface circuit ahead of the implementation phase by using the available design timing information. It is now possible to detect specification inconsistencies prior to the implementation of the circuit, and to guide the implementation phase (i.e. time-driven partitioning, placement, and routing). Finally the obtained bounds on the interface path delays can be used to verify that the final interface circuit satisfies the design constraints.

3. Protocol specification

Microprocessor components transfer information in the form of signals via 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 of the protocol. We use an abstraction of STG's called action graphs to capture the behavior of component interface protocols [7]. The design of the interface between two components is carried out by *merging* the components' corresponding action graphs. The resulting merged action graph can be transformed into a timed STG which is the starting point of our timing analysis. This section introduces our timed STG representation, affined to Vanbekbergen's timed STG's [12] and to Rokicki's orbital nets [11]. First we present the underlying timed Petri net.

3.1 Timed Petri net model

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

The set of places is partitioned into two subsets Pl_o and Pl_c. Time labels assigned to places belonging to Pl_o, the set of *operational places*, are used to model circuit delay. Time labels assigned to places belonging to Pl_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 subsets Pl_o and Pl_c, i.e. $F = F_o \cup F_c$, where $F_o \subseteq (Pl_o \times Tr) \cup (Tr \times Pl_o)$ and $F_c \subseteq (Pl_c \times Tr) \cup (Tr \times Pl_c)$.

The preset (postset) of a transition t is the set of incoming places to (outgoing from) t and is denoted $\bullet t$ ($t\bullet$). The intersection of $\bullet t$ (or $t\bullet$) with Pl_o is denoted $\bullet t_o$ (or $t\bullet_o$); likewise for $\bullet t_o$ and $t\bullet_o$. The firing rule for Petri nets is extended to take into consideration 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 must fire 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. Note the asymmetry between constraint places and operational places: only tokens in operational places are necessary to enable a transition; also tokens are perdurable in operational places but not in constraint places.

3.2 Ports, signals and signal transitions

Ports are designated with 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 the 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 binary signal transitions. A signal transition $(a, +)$ or in shorthand $a+$, indicates a positive transition of the signal value at the output port a. (A can be naturally extended to describe multi-valued signals [12], however for the purpose of our presentation we consider only binary transitions.)

3.3 Signal transition graphs

STG's are Petri nets whose transitions are interpreted as signal transitions. A timed STG is a triple $\langle$TPN, Y, $\Delta\rangle$ where *TPN* is a marked timed Petri net, Y is as set of signals, and Δ: $Tr \rightarrow A$ is a labelling function which assigns each $t \in Tr$ of the Petri net with a signal transition $a! \in A$.

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). Usually the *validity* of an STG is checked by ensuring that the corresponding state graph is consistent [12]. In the sequel we consider the subclass of STG's whose underlying Petri nets are marked graphs that satisfy:

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.

Definition 3.3.1.- A timed STG is time-consistent if no constraint place flags a violation during any possible execution of the STG.

4. Timing analysis

This section discusses the timing analysis of valid timed STG's. The first problem studied in this section is constraint satisfaction, i.e., to determine if a set of operational delays of a circuit satisfies the given timing constraints. The second problem considers a reverse form of the constraint satisfaction problem, namely given a set of known operational delays and timing constraints, determine possible values of variable operational 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 operational delays are the delays of the interface logic that is yet to be synthesized.

4.1 Time consistency

In this subsection we propose an optimization formulation to check for time-consistency of valid timed STG's. 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 $\otimes$ is defined by:

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

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

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 $\beta = [b_{\min}, b_{\max}]$.

In a marked graph, $|\bullet p| = |p\bullet| = 1$, thus places can be drawn as links between two transitions. To make our algebra more tractable, we have adopted the convention of denoting labels associated with constraint places using the symbol Δ while for the labels associated with operational places we use the symbol γ. Consider now transition d in Figure 1 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 Figure 1. According to the firing rule, d sees a token in a place 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 happens within the interval $max\,(\tau_a + \gamma_1, \tau_b + \gamma_2, t_c + \gamma_3)$. In this example τ_i are single-point intervals but in general they can be intervals too. If there is only one operational place in $\bullet d$, say the one labelled γ_1, the expression for τ_d becomes $\tau_a + \gamma_1$ as expected.

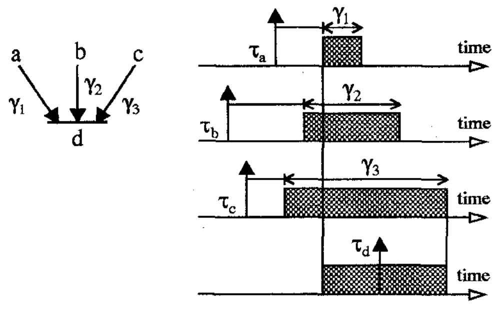

Figure 1. Firing of a transition.

A constraint place between two transitions a and b (see

Figure 2) signals a violation *iff* τ_b does not occur within the constraint interval after the occurrence of τ_a (shadowed area). A constraint place is said to be time-consistent if it does not signal a violation under any possible execution of the STG. We can determine if the place ever signals a violation under any possible execution of the STG if we know bounds on the time separation from τ_b to τ_a, written $\tau_b - \tau_a$. A constraint place is time-consistent *iff*

$$\tau_b - \tau_a \subseteq \Delta_1 \qquad \text{[Eq. 1]}$$

An STG is time-consistent *iff* all the constraint places are time-consistent. It is evident that to be able to calculate Eq. 1, the involved transitions must have a common ancestor, i.e., there must exist a transition x such that there is a path from x to a and from x to b, otherwise it could not be possible to find bounds on $\tau_b - \tau_a$ and the constraint could not be satisfied (unless $\Delta_1 = (-\infty, +\infty)$, the trivial constraint interval). We call such x a *fork transition* corresponding to transitions a and b if there are two lattices whose common least upper bound is x, and with greatest lower bounds a and b respectively. The significance of a fork transition is that τ_i in Eq. 1 can be computed relative to x.

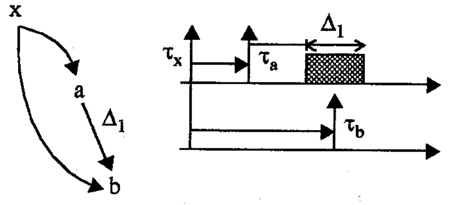

Figure 2. A constraint between two transitions.

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. We further assume that the execution of the graph results in periodic behavior so that the unfolded graph can be analyzed by looking only at a finite subgraph [9]. Figure 3 shows a simple protocol between two signals and its corresponding unfolded graph. In the initial marking (time zero) there are tokens in the places labelled with γ_4 and γ_5. (If initially the occurrences of $\underline{a}$+ and b+ did not follow γ_4 and γ_5 as shown in Figure 3, we could specify initial *reset* values γ_{4_0} and γ_{5_0}, equivalent to the reset rules in [9, 1]).

The time of occurrence of any event is computed as follows: starting from time zero in topological order we assign a time interval of occurrence to each transition in the graph. For example, the first occurrence of a+ is within γ_4 while the first occurrence of b+ is within max $(\gamma_4 + \gamma_1, \gamma_5)$. Note that in Figure 3 the subgraph between any two occurrences of b+ is repeated indefinitely. Then if we want to find the difference between the occurrences of two transitions in the same cycle, we do not need to start from time zero, but rather we can use b+ as a relative time origin. Transitions such as b+ are considered fork transition candidates to compute the time separation between events in a *per cycle* basis.

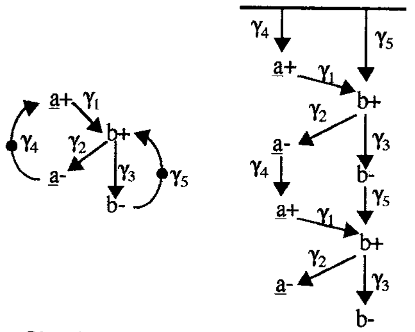

Figure 3. Simple signal transition graph and a partial view of its unfolded infinite acyclic graph.

Formally to find the time separation between two transitions a and b a fork transition x is identified such that there are two lattices in the unfolded graph starting from x and ending in a or b respectively. The time separation is computed as the interval difference between the occurrence times of b and a relative to x. For example the separation between transitions $b+^i$ and $\underline{a}+^i$ in Figure 3 for any cycle $i > 0$ ($i = 0$ is the initial cycle) 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 $b+^{i-1}$. The satisfaction of Eq. 1 involves solving a linear optimization program and it will be discussed in section 4.2.

4.2 Constraint satisfaction problem

The strength of the time separation procedure outlined in section 4.1 is that its formulation lends itself to further important extensions of timing analysis for synthesis, the subject of section 4.3. In this section we discuss a methodology to solve a set of interval expressions of the form of Eq. 1 using an optimization approach.

The constraint satisfaction of an STG is equivalent to checking its time consistency. Therefore an STG is time consistent *iff* all constraint equations of the form of Eq. 1 are satisfied. Eq. 1 involves the subtraction of interval expressions, each containing a possibly nested application of the *max* function on γ_i of the unfolded graph. Thus Eq. 1 is a nonlinear interval expression. Using an approach adapted from [8], we can solve the constraint satisfaction problem by solving first a finite set of subproblems. A subproblem is produced by choosing one of the terms from each of the *max* functions as winner. 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 possible subproblems.

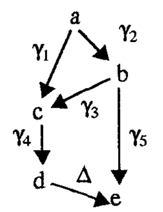

Figure 4. Constraint satisfaction.

Consider for example the graph shown in Figure 5. The constraint satisfaction equation is $\tau_e - \tau_d \subseteq \Delta$, where $\tau_e = max\ (\gamma_1, \gamma_2 + \gamma_3) + \gamma_4$, and $\tau_d = \gamma_2 + \gamma_5$ (*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:

min and max of $\{c_3 + c_4\} - c_5$
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 γ_1 = [0, 90], γ_2 = [0, 100], and $\gamma_3 = \gamma_4 = \gamma_5$ = [10, 20], the solution of $\tau_e - \tau_d$ is [0, 30]. Similarly for the subproblem $\gamma_1 \geq \gamma_2 + \gamma_3$, $\tau_e - \tau_d$ = [0, 100]. Thus for any Δ such that [0, 100] $\subseteq \Delta$, the constraint place of Figure 5 is time-consistent.

4.3 Timing analysis for synthesis

During synthesis it is often the case that not all the operational delays are known since the synthesis goal is precisely to generate a circuit. The constraint satisfaction procedure outlined in section 4.2 can be used for synthesis if conservative estimates for the unknown delays are used [9]. In this subsection we explain how to modify the constraint satisfaction procedure to find bounds on the unknown operational delays.

Assume that some of the operational intervals are unknown, denoted by δ_i. The constraint equations are now in terms of the known γ_i, the unknown δ_i, and the constraints Δ_i. As before we construct linear subproblems of the constraint satisfaction problem corresponding to a particular choice of a winner for each *max* term. For a given subproblem, a value *y* that satisfies the left-hand side of a constraint equation (i.e., $y \in \tau_b - \tau_a$) can be written as $y = f_a(c_i, d_i) - f_b(c_i, d_i)$, where f_a and f_b are two linear functions on variables c_i, d_i such that $c_i \in \gamma_i$ and $d_i \in \delta_i$. According to Eq. 1, $y \in \Delta$. Therefore the solution of a particular subproblem is the solution of the optimization program:

min $f_n\ (d_i)$
subject to
$y \in \Delta$, for all values $c_i \in \gamma_i$,
$d_i \geq 0$, and
conditions of the choice of max terms.

where f_n is the null function.

A particular solution consists of the set of feasible points $\{d_i\}$ which, when non-empty, is delimited by a (possibly unbounded) convex polytope. It is well known that every 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, it is only required to store additional direction vectors describing the edges to infinity). The total solution is the union of all the particular solutions.

For example let Δ be [0, 100] and δ_1 be unknown (the former γ_1) in the graph shown in Figure 5, with the other operational intervals unchanged. There are two subproblems, corresponding to $\delta_1 \leq \gamma_2 + \gamma_3$, and to $\delta_1 \geq \gamma_2 + \gamma_3$. The optimization program generated by the first subproblem is:

min $f_n\ (\gamma_i)$
subject to
$c_3 + c_4 - c_5 \in \Delta$, for all $c_i \in \gamma_i$, i = 2..5,
$d_1 \geq 0$, and
$d_1 - c_2 - c_3 \leq 0$.

The partial solutions of the two subproblems are [0, 10] and [0, 90] respectively, thus δ_1 = [0, 90].

5. Example: Synchronous memory read cycle

In this section we present a simple example that illustrates the type of analysis allowed by our procedure. Figure 6 shows a timed STG describing the interface between a CPU and a memory chip for the read operation.

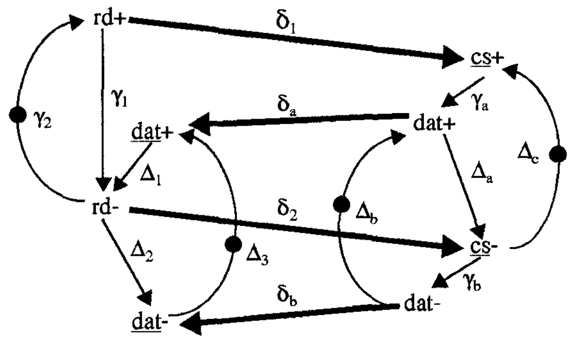

Figure 5. Synchronous memory read cycle.

We can identify two subgraphs: The valid graph shown at the left describes the CPU synchronous read protocol, i.e., the cycle completes after a delay specified by the CPU clock (represented by γ_1 and γ_2), while the right subgraph describes the memory protocol where the chip select signal *cs* controls the data; γ_a is the memory read access time. The thick links represent the interface: δ_1 and δ_2 describe delays through the selection logic; δ_a and δ_b correspond to the interface data path. Constraints Δ_1 and Δ_2 are respectively the setup and hold times of *dat* with respect to the read strobe *rd-*. Δ_a specifies that the memory chip should be deselected after the data have been accessed. The other constraints specify a causal ordering between the last transition of a cycle and its corresponding opposite transition in the next cycle.

Values for γ_i and Δ_i can be obtained from the manufacturer's manual sheets. Bounds on δ_i can be found by applying the procedure outlined in sections 4.2 and 4.3. To write the constraint equations for each constraint (Eq. 1), we use one period of the unfolded acyclic graph (see Figure 7). For example the constraint equation corresponding to Δ_2 is $\tau_{\underline{dat}-} - \tau_{rd-} = \delta_2 + \gamma_b + \delta_b \subseteq \Delta_2$.

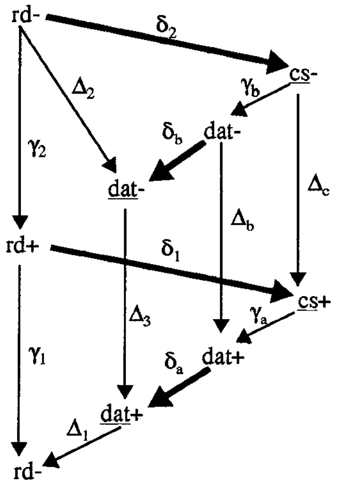

Figure 6. Unfolded graph showing one period of the memory read cycle.

For $\gamma_1 = [100, 200]$, $\gamma_2 = [10, 100]$, $\gamma_a = [20, 40]$, $\gamma_b = [22, 38]$, $\Delta_1 = [25, \infty)$, $\Delta_2 = [15, \infty)$, $\Delta_a = [10, \infty)$, and $\Delta_3 = \Delta_a = \Delta_b = [0, \infty)$, the solution is a four-dimensional polytope corresponding to the four δ_i variables. Figure 8 shows two 2-dimensional projections of the polytope. Using the convexity property of the polytope we can determine extreme values for the δ_i variables. For example, $\delta_1 \in [8, 35]$, and $\delta_2 \in [0, 27]$. Furthermore the projections also show the relationship that need be satisfied among the δ_i variables. For instance if due to a technology mapping decision it is determined that $\delta_1 \in [10, 15]$ then $\delta_a \in [0, 20]$; likewise δ_b and δ_2 would also be further constrained.

6. Conclusions

Traditionally during synthesis the timing analysis is carried out by iteratively estimating and adjusting the values of the unknown control and data path delays. In this paper we consider the design of interfaces in microprocessor-based systems. We presented a methodology that can be used to determine tight bounds on the interface path delays in advance of the implementation phase. This information can be used advantageously (i) to detect inconsistencies in the design before implementation, (ii) to guide lower synthesis stages, and (iii) to verify that the final implementation satisfies the requirements.

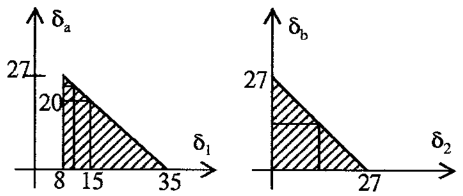

Figure 7. Projections of the solution polytope.

As important area of future work is finding efficient algorithms to carry out the timing analysis outlined in this paper.

Acknowledgments

We want to thank Tom Rokicki and Peter Vanbekbergen for constructive discussions. This research was supported in part by strategic NSERC grants and by IRIS. M. Escalante was also supported by a University of Victoria fellowship.

References

[1] T. Amon, H. Hulgaard, S. M. Burns, and G. Borriello, "An algorithm for exact bounds on the time separation of events in concurrent systems," in *Proc. ICCD*, pp. 166-173, 1993.

[2] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time petri nets," *IEEE Trans. on Software Engineering*, vol. 17, pp. 259-273, Mar. 1991.

[3] W. P. Birmingham, A. P. Gupta, and D. P. Siewiorek, "The MICON system for computer design," *IEEE Micro*, vol. 9, pp. 61-67, Oct. 1989.

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

[5] S. M. Burns, "Performance analysis and optimization of asynchronous circuits," PhD dissertation, Tech. Rep. Caltech-CS-TR-91-01, Dec. 1990.

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

[7] M. A. Escalante and N. J. Dimopoulos, "Timed asynchronous interface design in microprocessor-based systems," in *Proc. of the Canadian Conference on VLSI*, pp. 7-1-7-6, Nov. 1993.

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

[9] C. J. Myers and T. H.-Y. Meng, "Synthesis of timed asynchronous circuits," *IEEE Trans. on VLSI Systems*, vol. 1, pp. 106-119, June 1993.

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

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

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