*Sādhanā* Vol. 27, Part 2, April 2002, pp. 129–162. © Printed in India

# Efficient algorithms for approximate time separation of events[*]

SUPRATIK CHAKRABORTY[1], DAVID L DILL[2] and KENNETH Y YUN[3]

[1]Department of Computer Science and Engineering, Indian Institute of Technology – Bombay, Mumbai 400 076, India
[2]Computer Science Department, Stanford University, Stanford, CA 94305, USA
[3]Electrical and Computer Engineering Department, University of California, San Diego, La Jolla, CA 92093-0407, USA
e-mail: supratik@cse.iitb.ac.in; dill@cs.stanford.edu; kyy@paradise.ucsd.edu

**Abstract.** Finding bounds on time separation of events is a fundamental problem in the verification and analysis of asynchronous and concurrent systems. Unfortunately, even for systems without repeated events or choice, computing exact bounds on time separation of events is an intractable problem when both min and max type timing constraints are present. In this paper, we describe a method for approximating min and max type constraints, and develop a polynomial-time algorithm for computing approximate time separation bounds in choice-free systems without repeated events. Next, we develop a pseudo-polynomial time technique for analysing a class of asynchronous systems in which events repeat over time. Unlike earlier works, our algorithms can analyse systems with both min and max type timing constraints efficiently. Although the computed bounds are conservative in the worst-case, experimental results indicate that they are fairly accurate in practice. We present formal proofs of correctness of our algorithms, and demonstrate their efficiency and accuracy by applying them to a suite of benchmarks. A complete asynchronous chip has been modelled and analysed using the proposed technique, revealing potential timing problems (already known to designers) in the datapath design.

**Keywords.** Asynchronous systems; timing analysis and verification; approximate algorithms; convex approximation; time separation of events; bounded delay timing analysis.

## 1. Introduction

The behaviour of asynchronous and concurrent systems is naturally described in terms of events and their interactions. A fundamental problem in analysing such systems is to

[*]Parts of this work are based on "Timing Analysis of Asynchronous Systems using Time Separation of Events" by Chakraborty, Dill and Yun 1999 *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 18(8): 1061–1076 ©IEEE. Permission to publish this material has been obtained from IEEE

determine bounds of the time separation of events. Stated informally, we seek answers to questions such as: "How late can event $A$ occur after event $B$?" for arbitrary events $A$ and $B$. Time separation bounds find use in a wide variety of applications, including interface timing verification (Borriello 1988; Gahlinger 1990; Brzozowski *et al* 1991; McMillan & Dill 1992; Walkup 1995; Yen *et al* 1998), synthesis, optimization and verification of asynchronous circuits (Vanbekbergen *et al* 1992; Amon 1993; Lavagno & Sangiovanni-Vincentelli 1994; Myers 1995; Hulgaard 1995; Chakraborty *et al* 1999), and performance analysis and scheduling of concurrent systems (Chou & Borriello 1995; Hulgaard 1995; Yen & Wolf 1995; Chakraborty *et al* 1999). In addition, it also has applications in the synchronous domain, e.g., optimal clock scheduling in circuits with latches (Sakallah *et al* 1990). The problem of computing time separation bounds is compounded in practice by statistical variations in manufacturing and operating conditions that introduce uncertainties in component delays. Consequently, finding bounds of time separation of events in the presence of uncertain component delays is an important practical problem.

Unfortunately, finding exact time separation bounds in systems with uncertain delays is a computationally intractable problem in general (McMillan & Dill 1992). This motivates the investigation of *efficient* techniques for computing bounds that are *approximate* in the worst-case, but fairly accurate in practice. In this paper, we address this problem in two parts. First, we describe a polynomial-time algorithm for computing approximate time separation bounds in systems without repeated events. Next, we use this algorithm to design a pseudo-polynomial-time algorithm for analysing a class of systems with repeated events. All bounds computed by our algorithms are provably conservative. Nevertheless, experiments indicate that they are fairly accurate in practice. For efficiency, we restrict our analysis to systems without choice or conditional behavior. While this is a significant restriction, empirical evidence suggests that several important applications can be modelled as choice-free systems and analysed by our technique. We demonstrate the effectiveness of our approach by verifying interface timing constraints and by computing performance metrics of an asynchronous chip design (Yun *et al* 1998).

Several temporal logics described in the literature can also be used to specify properties expressible as time separation between events (see Alur & Henzinger 1992 for a survey). Typically, the specification is given as a formula in a suitable temporal logic, and the system under verification is specified using one of several formalisms, e.g. timed transition systems, temporal logic formulae, etc. Depending on the formalisms used to specify the system and the property, deductive or algorithmic techniques (like model checking) can be used to verify that the system satisfies the specification. Time separation of events analysis, on the other hand, starts from an event-based description of the system and computes bounds of all relevant time separation of events. Comparison of the computed bounds with required bounds then reveals whether specified timing constraints (properties) are violated. Clearly, this approach restricts the type of properties one can specify and verify to only those expressible as time separation between events. In contrast, temporal logics like TPTL, RTTL, RTCTL, TCTL etc. (Alur & Henzinger 1992) are significantly richer in their expressive power. Nevertheless, the restriction allows us to design efficient algorithms with a high degree of accuracy for an important class of problems that arise in the analysis of asynchronous and concurrent systems. In addition, the computed time separation bounds also allow us to compute system performance metrics without any additional effort.

The remainder of this paper is organised as follows. Section 2 describes a formalism for representing timing constraints between events, and characterizes a class of systems called *tightly-coupled systems*. Section 3 describes related work on time separation of events analysis. In § 4, we describe an approximation technique for min and max timing constraints, and use this to design a polynomial-time algorithm for computing bounds on time separation of events in systems without repeated events. Section 5 describes an algorithm for computing approximate bounds of time separation of events in tightly-coupled systems, while § 6 describes application of this algorithm to an asynchronous chip design. Section 7 explains the drawbacks and limitations of our approach. Finally, we conclude the paper in § 8.

## 2. Problem representation and formalization

In this section, we describe a formalism for representing timing constraints between events. We then characterize a class of choice-free systems with repeated events, and formalize the time separation of events problem for such systems.

### 2.1 *Timing constraint graph*

Following existing convention (Borriello 1988; Brzozowski *et al* 1991; Burns 1991; McMillan & Dill 1992; Vanbekbergen *et al* 1992), we represent timing constraints between events by a directed, labeled graph, $G = (V, E)$, called the *timing constraint graph*. Vertices in $V$ represent events, and edges in $E$ represent timing constraints between events. In the following, we will use the terms *vertices* and *events* interchangeably. We use acyclic graphs to model systems without repeated events, and add cycles in the graphs to model events that repeat over time.

Every event $v$ in a timing constraint graph is associated with a min or max *type* that specifies the nature of timing constraint associated with the event. An event with no incoming edges is called a *source* event, while one with no outgoing edges is called a *sink* event. Every edge $\langle u, v\rangle$, from $u$ to $v$, is labeled with a delay $\delta_{u,v}$ that represents the delay in the propagation of event $u$'s effect to the component that generates event $v$. The edge delays are constrained to lie within fixed lower and upper bounds, $d_{u,v}$ and $D_{u,v}$, i.e., $0 \leq d_{u,v} \leq \delta_{u,v} \leq D_{u,v}$. For notational convenience, we represent edge delays as $[d_{u,v}, D_{u,v}]$.

In order to model systems with repeated events, we add cycles in timing constraint graphs. Thus, a general timing constraint graph has two components: a finite acyclic component

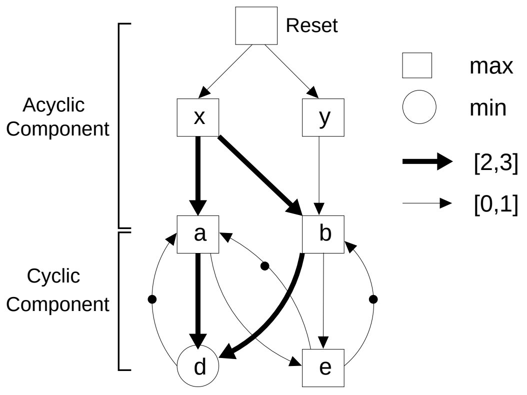


**Figure 1.** A cyclic timing constraint graph. Square vertices represent max events, circles represent min events. Edge delays are shown in the legend.

modelling the behaviour of the system immediately after it is started, and a finite cyclic component modelling the subsequent behaviour that repeats over time (see figure 1). A vertex in the acyclic component represents a single occurrence of an event, whereas a vertex in the cyclic component represents infinite occurrences of an event. The notation of Amon and coworkers (Amon *et al* 1992; Hulgaard *et al* 1995) is used to distinguish between different occurrences of the same event: the $k$th occurrence of event $v$ is denoted $v_k$, and $k$ is called the *occurrence index* of $v_k$. By convention, the first occurrence of $v$ is denoted $v_1$. In addition, it is assumed that there exists a unique Reset event with no incoming edges that represents the start of operation of the system.

Every edge in a timing constraint graph is designated as either *marked* or *unmarked*. A marked edge from $u$ to $v$ represents a dependency of the time of occurrence of event $v_{k+1}$ on that of event $u_k$, for all occurrence indices $k$. On the other hand, an unmarked edge from $u$ to $v$ denotes a dependence of the time of occurrence of $v_k$ on that of $u_k$. A marked edge is depicted by an edge with a $\bullet$ as shown in figure 1. For clarity, edges with different delay intervals are represented by different arrow types in this paper (see, for example, the legend in figure 1).

Let $\tau_{v_k}$ denote the time of occurrence of event $v_k$ and $preds(v_k)$ denote the set of events with an edge to $v_k$, i.e., predecessors of $v_k$ in the graph. If $v_k$ is of max type, then $\tau_{v_k}$ is given by $\max_{u \in preds(v_k)}(\tau_u + \delta_{u,v_k})$ where $d_{u,v_k} \leq \delta_{u,v_k} \leq D_{u,v_k}$. For brevity, we will henceforth write this as $\tau_{v_k} = \max_{u \in preds(v_k)}(\tau_u + [d_{u,v_k}, D_{u,v_k}])$. For a min type event, the expression for $\tau_{v_k}$ is similar, with max replaced by min.

As an example, the graph of figure 1 represents the following set of timing constraints:

$$\begin{aligned}
&\tau_{x_1} = \tau_{\mathsf{Reset}_1} + [0, 1]; \quad \tau_{y_1} = \tau_{\mathsf{Reset}_1} + [0, 1];\\
&\tau_{a_1} = \tau_{x_1} + [2, 3]; \qquad \tau_{b_1} = \max(\tau_{x_1} + [2, 3], \tau_{y_1} + [0, 1]);\\
&\forall k \geq 1; \quad \tau_{d_k} = \min(\tau_{a_k} + [2, 3], \tau_{b_k} + [2, 3]);\\
&\forall k \geq 1; \quad \tau_{e_k} = \max(\tau_{a_k} + [0, 1], \tau_{b_k} + [0, 1]);\\
&\forall k \geq 1; \quad \tau_{a_{k+1}} = \max(\tau_{d_k} + [0, 1], \tau_{e_k} + [0, 1]);\\
&\forall k \geq 1; \quad \tau_{b_{k+1}} = \tau_{e_k} + [0, 1].
\end{aligned}$$

Let $\mathcal{T}$ denote the set of all time variables in the system, i.e., $\mathcal{T} = \{\tau_v : v \in V\}$. Let $\Re^+$ be the set of non-negative real numbers. A *consistent timing assignment*, $TA : \mathcal{T} \to \Re^+$, is an assignment of non-negative reals to the time variables, such that all constraints in the timing constraint graph are satisfied.

Following the terminology of Amon and coworkers (Amon *et al* 1992; Hulgaard *et al* 1995) and Myers & Meng (1993), a timing constraint graph is said to be *well-formed* if

(1) there are no cyclic components, or
(2) there are one or more cyclic components with the following properties: (a) every cycle has at least one marked edge, and (b) for every event $v$ on a cycle, there exists at least one cycle with exactly one marked edge that contains $v$.

Given a well-formed graph, if we assign every edge a delay within the corresponding bounds, the time of occurrence of every event is well-defined. Specifically, for graphs with cyclic components, condition 2a ensures that no event is deadlocked, while condition 2b ensures that the time of occurrence of $v_k$ is well-defined for every occurrence index $k$. All graphs considered in this paper are assumed to be finite and well-formed.

A graph $G = (V, E)$ with one or more cycles represents timing constraints between an infinite set of event occurrences. These constraints can be equivalently represented by an

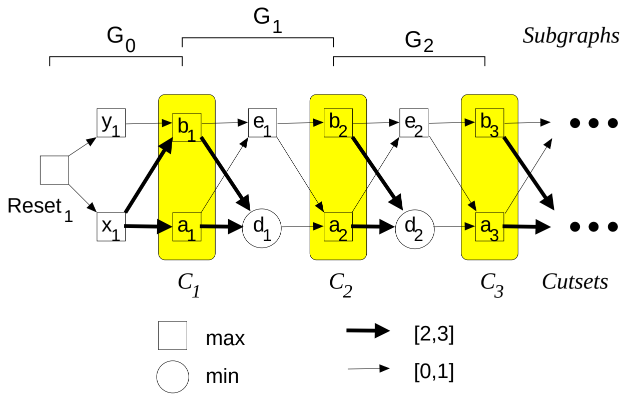


**Figure 2.** Unfolded version of cyclic timing constraint graph in figure 1.

*infinite acyclic* graph $G^* = (V^*, E^*)$ constructed as follows. For every event $v$ in the acyclic component of $G$, we create a corresponding event $v_1$ in $G^*$. For every event $v$ in the cyclic component of $G$, we create an infinite set of events in $G^*$, namely $\{v_i : i \geq 1\}$. Each such $v_i$ in $G^*$ is associated with the same operator (min or max) as $v$ is in $G$. Whenever there exists a marked edge from $u$ to $v$ in $G$, edges are drawn from $u_i$ to $v_{i+1}$ for all occurrence indices $i$ in $G^*$. For every unmarked edge from $u$ to $v$ in $G$, edges are drawn from $u_i$ to $v_i$ in $G^*$. The edges thus drawn in $G^*$ are labeled with the same delay interval as the edge from $u$ to $v$ in $G$. We call $G^*$ the *unfolded graph* of $G$. As an example, figure 2 (ignoring the enclosing boxes $b_1, a_1, b_2, a_2$ etc. and the annotations regarding subgraphs and cutsets) depicts the initial portion of the infinite acyclic graph obtained by unfolding the timing constraint graph in figure 1.

## 2.2 *Tightly-coupled systems*

Given an unfolded graph, $G^* = (V^*, E^*)$, a path from $u$ to $v$ is defined as a sequence of events $(u, \ldots x, y, \ldots v)$ such that $\langle x, y\rangle \in E^*$ for every pair of consecutive events $x$ and $y$. By definition, $(u)$ is a path from event $u$ to itself. We say that $v$ is *reachable* from $u$ if there exists a path from $u$ to $v$; otherwise, $v$ is *unreachable* from $u$.

Given a set, $X$, of events, the set of events reachable from some $v$ in $X$ is denoted $\mathcal{R}(X)$. The set of events unreachable from all events in $X$ is denoted $\mathcal{R}'(X)$. A *cutset*, $C$, is a finite set of events such that every path from Reset to every event reachable from $C$ passes through at least one event in $C$. For example, the set $\{a_1, b_1\}$ in figure 2 is a cutset of the infinite unfolded graph.

DEFINITION 1

*A timing constraint graph with cycles is said to be* ***tightly-coupled*** *if the unfolded graph has a sequence of cutsets, $C_i$ for all $i \geq 1$, such that:*

**P1:** *There are finitely many events that are* not *reachable from $C_1$.*
**P2:** *There exists a finite, positive integer $\alpha$ such that $C_i = \{v_{k+(i-1)\alpha} : v_k \in C_1\}$.*
**P3:** *For all $i$ different from $j$, $C_i$ is disjoint from $C_j$.*
**P4:** *For every $u$ in $C_i$ and every $v$ in $C_{i+1}$, there exists at least one path from $u$ to $v$, such that all events along the path are of max type.*

As an example, the sets of events enclosed within boxes $C_1$, $C_2$ and $C_3$ in figure 2 represent the first three cutsets in an infinite sequence satisfying properties **P1** through **P4**. Therefore, the graph in figure 1 is tightly-coupled.

The reader may consider property **P4** overly restrictive and wonder if removing min type events, such as $d_i$ in figure 2, alters the temporal behaviour of the system. However, it can be shown that removing the $d_i$'s in figure 2 indeed alters the maximum possible time separation from $b_i$ to $a_i$, for all $i$ greater than or equal to 2.

Let $G_i = (V_i, E_i)$ denote the subgraph "sandwiched" between cutsets $C_i$ and $C_{i+1}$, for all $i$ greater than or equal to 1. Formally, $V_i = \mathcal{R}(C_i) \cap (\mathcal{R}'(C_{i+1}) \cup C_{i+1})$ and $E_i = \{\langle u, v\rangle : u, v \in V_i \text{ and } \langle u, v\rangle \in E^*\}$. For the sake of completeness, subgraph $G_0 = (V_0, E_0)$ is defined as the component modelling the behaviour of the system from Reset to the events in $C_1$, i.e., $V_0 = \mathcal{R}'(C_1) \cup C_1$ and $E_0 = \{\langle u, v\rangle : u, v \in V_0 \text{ and } \langle u, v\rangle \in E^*\}$. Due to the repetitive structure of the unfolded graph, it is easy to see that $G_i$ is *isomorphic* to $G_1$, for all $i$ greater than or equal to 1. For a given choice of cutsets, the behaviour modelled by $G_0$ is referred to as the *initial behaviour* of the system, and the behaviour modelled by $G_i$ $(i \geq 1)$ is referred to as *iteration i* of the system.

If $\alpha$ in property **P2** is chosen greater than 1 (see, for example, figure 8b below), subgraph $G_i$ may include multiple occurrences of the same event. To keep event labels unambiguous and consistent, we relabel events after unfolding a graph as follows:

- All events in $G_0$ and $G_1$, except those in cutset $C_2$, are assigned unique labels with occurrence index 1.
- For every event, $v_1$, in $G_1$, the corresponding event in $G_i$ (correspondence defined by the isomorphism from $G_1$ to $G_i$) is labeled $v_i$. This, of course, implies that $C_i = \{v_i : v_1 \in C_1\}$, so $\alpha$ in property **P2** reduces to 1.

### 2.3 *The problem*

The problem we wish to address can now be stated as follows. Given a tightly coupled system with subgraphs $G_i$ $(i \geq 0)$ of the unfolded graph,

(1) Determine upper bounds of time separations of all ordered pairs of events in $G_0$.
(2) Determine upper bounds of time separations of all ordered pairs of events in $G_i$, maximised over all $i \geq 1$.

Although both upper and lower bounds of time separations are of interest to us, a lower bound of $(\tau_v - \tau_u)$ can be obtained by reversing the sign of the upper bound of $(\tau_u - \tau_v)$. Therefore, it suffices to compute upper bounds for every *ordered pair* of events.

## 3. Related work

Borriello (1988) modelled the timing of interface signals using linear inequalities, and proposed an exact timing verification algorithm based on computing shortest paths between all pairs of vertices in a graph. Gahlinger (1990), Brzozowski *et al* (1991) Mavaddat & Gahlinger (1998) also used linear constraints to verify the consistency of interface timing specifications and to check the satisfiability of timing requirements.

McMillan & Dill (1992) showed that computing exact bounds of time separation of events is NP-complete even for acyclic graphs when both min and max type constraints are present. They proposed a polynomial-time algorithm for systems with only max constraints, and a pseudo-polynomial time algorithm for systems with max and linear constraints. They also described a branch-and-bound technique, with complexity exponential in the number of min type events, for analysing systems with both min and max constraints. Vanbekbergen *et al* (1992) proposed a polynomial-time algorithm for computing bounds in acyclic graphs with only max constraints. Burks & Sakallah (1993) posed the time separations problem for acyclic graphs as a *min-max linear program* and showed it to be NP-complete. They proposed a branch-and-bound technique and a mixed integer linear program formulation to solve the problem. Unfortunately, both their techniques have worst-case exponential complexity.

Walkup and others (Walkup & Borriello 1994; Walkup 1995) proposed the ShortCircuit algorithm for analysing systems of max and linear constraints, with applications to interface timing verification and interface logic synthesis. Subsequently, Yen *et al* (1998) proposed the MaxSeparation algorithm for solving the same problem. Both these algorithms analyse systems without repeated events and are *conjectured* to run in polynomial-time. The exact complexity of computing time separation bounds in systems with max and linear constraints is, however, still an open question.

Amon & Borriello (1992) proposed a technique based on *constraint logic programming* (CLP) for timing analysis of systems with min, max and linear constraints. Subsequently, Girodias *et al* (1997) have also described a method based on CLP augmented with relational interval arithmetic, for analysing similar systems. Although these techniques are exact, they apply to systems without repeated events, and have worst-case exponential complexity.

Burns (1991) and Lee (1995) proposed techniques for determining the cycle time of asynchronous systems with repeated events. Burns' technique applies to max-only systems and has polynomial time complexity, while Lee's technique applies to more general systems at the cost of worst-case exponential complexity. Gunawardena (1994) gave a theoretical framework for analysing cyclic systems of min and max constraints with fixed delays. He studied the periodic behaviour of such systems and gave an analytic formula for the period. Several other researchers have also modelled asynchronous circuits as systems of repeated events with different delay assumptions, and proposed techniques for computing their performance metrics (Kudva *et al* 1994; Nielson & Kishinevsky 1994; Williams 1994; Tofts 1995; Xie & Beerel 1997; Ebergen & Berks 1997; Berks & Ebergen 1997).

Myers & Meng (1993) described a polynomial-time algorithm for approximate timing analysis of max-only systems with repeated events and bounded delays. In their method, a cyclic graph is used to represent timing constraints between events. Their algorithm effectively unfolds this graph and analyses a finite subgraph of the unfolded graph to compute approximate bounds of time separation between events. Our approach differs from theirs in two important respects: (a) we propose a different algorithm for computing approximate bounds in acyclic graphs, enabling us to analyse systems with both min and max constraints, and (b) we define a notion of convergence and provide an analytic upper bound of the number of iterations needed for our iterative algorithm to converge.

Amon and coworkers (Amon *et al* 1992; Hulgaard *et al* 1995) also considered systems of repeated events with only max constraints, and proposed an algebraic technique for computing exact bounds of the time separation of events. Their technique implicitly unfolds a cyclic graph into an infinite acyclic graph and uses algebraic techniques to determine the maximum time separation between a pair of events, maximised over all unfoldings. Hulgaard and others (Hulgaard & Burns 1994; Hulgaard 1995) also extended this work to systems with certain types of choices. Recently, symbolic timing verification has been studied by Amon and others (Amon *et al* 1997; Amon & Hulgaard 1999). Timing verification using difference decision diagrams has also been attempted by Mller *et al* (1999). All these techniques, however, have worst-case complexity that is at least exponential in the size of the system description.

## 4. Analysing acyclic graphs

In this section, we develop a polynomial-time algorithm for computing approximate bounds on time separations of all pairs of events in an acyclic timing constraint graph. This algorithm is then used in § 5 to develop a pseudo-polynomial time algorithm for analysing tightly-coupled systems.

### 4.1 *An approximation strategy*

We start by approximating min and max constraints using systems of linear inequalities. Consider a max constraint of the form:

$$\begin{aligned} \tau_f &= \max(\tau_i + \delta_{i,f}, \ \ldots \ \tau_k + \delta_{k,f}), \quad \text{where} \\ & d_{i,f} \le \delta_{i,f} \le D_{i,f}, \\ & \qquad \vdots \\ & d_{k,f} \le \delta_{k,f} \le D_{k,f}. \end{aligned} \tag{1}$$

Mathematically, this is equivalent to:

$$\begin{array}{lc} (\tau_f \ge \tau_i + d_{i,f}) & \wedge \\ \qquad \vdots & \wedge \\ (\tau_f \ge \tau_k + d_{k,f}) & \wedge \end{array}$$
$$\begin{pmatrix} (\tau_f \le \tau_i + D_{i,f}) & \vee \\ \vdots & \vee \\ (\tau_f \le \tau_k + D_{k,f}) & \end{pmatrix}. \tag{2}$$

Assuming that the number of time variables is $n$, the *feasible region* of the above system of inequalities is the set of points $(\tau_f, \tau_i \ldots, \tau_k)$ in $n$-dimensional real space that satisfy the system of inequalities. It is well-known that the feasible region of the conjunction of linear inequalities in $n$ variables is a convex polytope in $n$-dimensional real space. However, the disjunctions in (2) cause the feasible region of (2) to be non-convex. As a consequence, efficient convex optimization techniques, like linear programming or the shortest path algorithm cannot be applied to determine bounds of the difference between time variables.

We, therefore, intend to approximate the non-convex feasible region by a *convex over-approximation* or *envelope*, and compute bounds of the difference between time variables within the convex envelope. Since the original feasible region is contained in the envelope, maximizing the difference between a pair of variables in the envelope always gives an upper bound of the maximum separation in the original feasible space. Since the convex hull of $k$ points is the smallest convex envelope containing all the points, using the convex hull of the corners of the feasible region as the required envelope would give us the best approximation results. Unfortunately, computing the convex-hull of an $n$-dimensional region has worst-case complexity exponential in $n$. Consequently, it is necessary to develop more efficient techniques for computing a reasonably small convex envelope of the feasible region.

In order to obtain the required convex envelope, we propose to approximate each max (and min) constraint, as in (1), by conjunctions of linear inequalities. A naive way to do this would be to retain all conjuncts in (2) of the form $\tau_f \geq \tau_i + d_{i,f}$ and to discard the disjunction of linear inequalities. However, this leads to a crude over-approximation since the resulting system of inequalities does not constrain the maximum value of $\tau_f$. Hence, for the approximation to be good, the disjunction of inequalities in (2) must not be ignored. Our solution, therefore, is to derive a system of inequalities from (2), in which only the conjunction operator is used.

Let $s$ be an arbitrary event in the system. In the following, we wish to obtain bounds of the time separation between $s$ and $f$, where $f$ is a max-type event, as in constraint (1). For every predecessor $i$ of $f$, let $\Delta(i, s)$ and $\Delta(s, i)$ be *any* real numbers satisfying the following inequalities:

$$\tau_s \leq \tau_i + \Delta(i, s), \tag{3}$$

$$\tau_i \leq \tau_s + \Delta(s, i). \tag{4}$$

The values of $\Delta(s, i)$ and $\Delta(i, s)$ can now be used in conjunction with inequalities (2) to derive linear relations between the time of occurrence of $f$ and that of the arbitrary event $s$. Specifically,

$$\begin{aligned} \tau_f &\geq \tau_i + d_{i,f}, && \text{from (2), and} \\ \tau_i &\geq \tau_s - \Delta(i, s), && \text{from (3), imply that} \\ \tau_f &\geq \tau_s - \Delta(i, s) + d_{i,f}. \end{aligned}$$

Similarly,

$$\begin{aligned} \tau_f &\leq \tau_i + D_{i,f}, && \text{from (2), and} \\ \tau_i &\leq \tau_s + \Delta(s, i), && \text{from (4) imply that} \\ \tau_f &\leq \tau_s + \Delta(s, i) + D_{i,f}. \end{aligned}$$

Therefore, the inequalities in (2) imply the following:

$$\begin{array}{c} (\tau_f \;\geq\; \tau_s - \Delta(i, s) + d_{i,f}) \quad \wedge \\ \vdots \quad \wedge \\ (\tau_f \;\geq\; \tau_s - \Delta(k, s) + d_{k,f}) \quad \wedge \\ \left( \begin{array}{c} (\tau_f \;\leq\; \tau_s + \Delta(s, i) + D_{i,f}) \quad \vee \\ \vdots \quad \vee \\ (\tau_f \;\leq\; \tau_s + \Delta(s, k) + D_{k,f}), \end{array} \right). \end{array} \tag{5}$$

The conjunctions in (5) simplify to

$$\tau_f \geq \tau_s + \max_{l \,\in\, preds(f)} (-\Delta(l, s) + d_{l,f}), \tag{6}$$

and the disjunctions are equivalent to

$$\tau_f \leq \tau_s + \max_{l \in preds(f)} (\Delta(s,l) + D_{l,f}). \tag{7}$$

Since choosing $s$ to be the same as $f$ in the above inequalities is uninteresting, it is henceforth assumed that $s$ is not equal to $f$. Thus, we obtain the following from inequalities (6) and (7):

$$\begin{aligned} &\forall s \neq f, \\ -&\max_{l \in preds(f)} (\Delta(s,l) + D_{l,f}) \leq \tau_s - \tau_f, \text{ and} \\ &\tau_s - \tau_f \leq \min_{l \in preds(f)} (\Delta(l,s) - d_{l,f}). \end{aligned} \tag{8}$$

The system of inequalities (8) is now used as an approximation of the max constraint in (1), and inequalities (2) are discarded. The following important observation follows from inequalities (8).

*Observation* 1. Let $f$ be a max-type event, $s$ an arbitrary event different from $f$ and $\Delta(s,l)$ and $-\Delta(l,s)$ upper and lower bounds respectively of $(\tau_l - \tau_s)$, for each predecessor $l$ of $f$. Then, the value of $(\tau_s - \tau_f)$ is bounded above by $\min_{l \in preds(f)}(\Delta(l,s) - d_{l,f})$ for all consistent timing assignments. Similarly, the value of $(\tau_f - \tau_s)$ is bounded above by $\max_{l \in preds(f)} (\Delta(s,l) + D_{l,f})$ for all consistent timing assignments.

Similar considerations for a min constraint lead to the following observation.

*Observation* 2. Let $f$ be a min-type event, $s$ an arbitrary event different from $f$ and $\Delta(s,l)$ and $-\Delta(l,s)$ upper and lower bounds respectively of $(\tau_l - \tau_s)$, for each predecessor $l$ of $f$. Then, the value of $(\tau_f - \tau_s)$ is bounded above by $\min_{l \in preds(f)}(\Delta(s,l) + D_{l,f})$ for all consistent timing assignments. Similarly, the value of $(\tau_s - \tau_f)$ is bounded above by $\max_{l \in preds(f)} (\Delta(l,s) - d_{l,f})$ for all consistent timing assignments.

*Example* 1. Consider the following system of timing constraints:

$$\begin{aligned} \tau_3 &= \max(\tau_1 + \delta_{1,3},\ \tau_2 + \delta_{2,3}), \\ 1 &\leq \delta_{1,3}, \delta_{2,3} \leq 3, \\ \Delta(1,2) &= \Delta(2,1) = 1. \end{aligned}$$

Figure 3a shows the feasible region of this system. The space enclosed within the “V”-shaped volume and extending infinitely in both directions along the ridge of the V represents the feasible region. The approximation strategy described above gives the following linear inequalities for this system:

$$\begin{aligned} &\tau_3 \geq \tau_1 + 1; \quad \tau_3 \leq \tau_1 + 4; \quad \tau_3 \geq \tau_2 + 1; \\ &\tau_3 \leq \tau_2 + 4; \quad -1 \leq \tau_2 - \tau_1 \leq 1 \quad . \end{aligned}$$

The feasible region of this system of inequalities is shown in figure 3b. Clearly, this forms a convex envelop of the original non-convex feasible region.

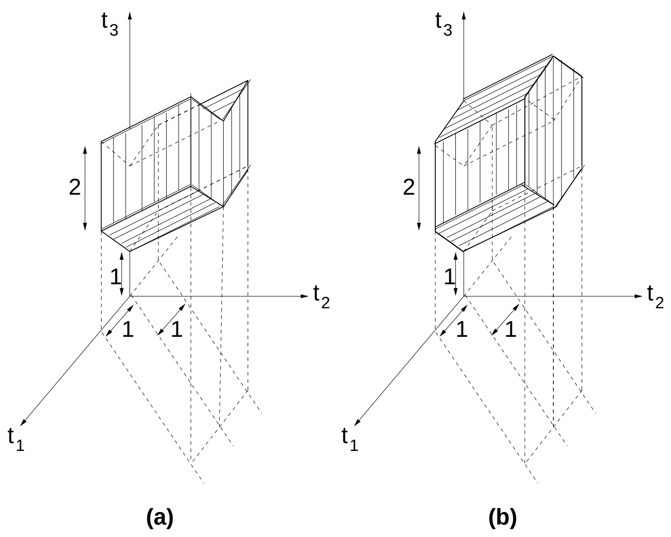


**Figure 3.** Illustrating the inclusion of the original feasible space in the approximate convex space.

## 4.2 *A polynomial-time algorithm*

The observations made above are now used to design a polynomial-time algorithm for computing approximate bounds of the time separation of all pairs of events in an acyclic graph. The algorithm accepts as input a finite, acyclic timing constraint graph with $n$ events. For graphs with multiple source events, the minimum and maximum time separations between all pairs of source events are also taken as inputs. The output of the algorithm is an $n \times n$ matrix, $\Delta$; the entry in row $i$ and column $j$, denoted $\Delta(i, j)$, is an upper bound of $(\tau_j - \tau_i)$.

It is assumed that the events in the timing constraint graph are topologically sorted and every event has a topological index greater than those of its predecessors. For a graph with $m$ source events, the source events are assumed to have indices 0 through $m - 1$.

In order to explain the way in which the algorithm works, we define *layer $i$* of the $\Delta$ matrix as the set of all $\Delta(j, i)$ and $\Delta(i, j)$ entries, with $j$ lying in the range 0 through $i$. The $\Delta$ matrix is viewed as made up of $n$ layers, numbered 0 through $n - 1$, as shown in figure 4 for a $4 \times 4$ matrix. The proposed algorithm, called AcyclicApproxSep, computes values of the entries in the $\Delta$ matrix in order of increasing layers. It starts with layer 0, i.e., $\Delta(0, 0)$, which has the value 0 trivially. It then computes the entries in layer 1, followed by the entries in layer 2, and so on, until all $n$ layers have been computed.

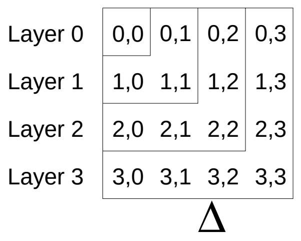


**Figure 4.** Layer wise construction of the $\Delta$ matrix.

```
/* Source events have indices 0 through $m-1$, other events have indices $m$ through $n-1$.*/
/* $preds(i)$ = set of events with an edge to event $i$; $\Delta$: an $n \times n$ matrix; $\Delta(i,j)$ = upper */
/* bound of $\tau_j - \tau_i$.                                                              */
/* Bounds of the time separations between all pairs of source events are given.            */
/* Algorithm computes conservative bounds on the time separations between all other pairs*/
/* of events.                                                                               */

AcyclicApproxSep ($G$ : acyclic graph; $\Delta$ : matrix)
InitializeDelta($\Delta$);
for each $i$ in $m$ to $(n-1)$
  1. ComputeInitialEstimates ($i$, $\Delta$);
  2. for each $j$ in 0 to $i-1$
       if (event $j$ is max type)
         (2a) $\Delta(i,j) =$
                $\min(\Delta(i,j),\ \max_{k \in preds(j)}(\Delta(i,k) + D_{k,j}))$;
         (2b) $\Delta(j,i) =$
                $\min(\Delta(j,i),\ \min_{k \in preds(j)}(\Delta(k,i) - d_{k,j}))$;
       else if (event $j$ is min type)
         (2c) $\Delta(i,j) =$
                $\min(\Delta(i,j),\ \min_{k \in preds(j)}(\Delta(i,k) + D_{k,j}))$;
         (2d) $\Delta(j,i) =$
                $\min(\Delta(j,i),\ \max_{k \in preds(j)}(\Delta(k,i) - d_{k,j}))$;

InitializeDelta ($\Delta$ : matrix)
 for each $i$ in 0 to $(n-1)$
   for each $j$ in 0 to $(n-1)$
     if $(i == j)$    $\Delta(i,i) = 0$;
     else if $(i < m$ AND $j < m)$      /* Source events */
       $\Delta(i,j)$ = given upper bound of $(\tau_j - \tau_i)$;
     else    $\Delta(i,j) = +\infty$;

ComputeInitialEstimates ($i$ : index; $\Delta$ : matrix)
 if (event $i$ is max type)
   for each $j$ in 0 to $(i-1)$
     (i) $\Delta(i,j) = \min_{k \in preds(i)}(\Delta(k,j) - d_{k,i})$;
     (ii) $\Delta(j,i) = \max_{k \in preds(i)}(\Delta(j,k) + D_{k,i})$;
 else if  (event $i$ is min type)
   for each $j$ in 0 to $(i-1)$
     (iii) $\Delta(i,j) = \max_{k \in preds(i)}(\Delta(k,j) - d_{k,i})$;
     (iv) $\Delta(j,i) = \min_{k \in preds(i)}(\Delta(j,k) + D_{k,i})$;
```

**Figure 5.** Algorithm for computing time separation of events in acyclic graphs.

Figure 5 shows the steps in the timing analysis algorithm[1]. After initialization, the algorithm computes the entries of the matrix from layer $m$ onwards. For each layer $i$, the algorithm first invokes function ComputeInitialEstimates to compute initial estimates of the entries in layer $i$ using observations 1 and 2 of § 4.1. The type of event $i$ is used to determine the expressions for the initial estimates. Referring to the pseudo-code in figure 5, note that the value of $j$ in steps (i) through (iv) of function ComputeInitialEstimates is less than $i$. Moreover, all

[1]This is an improved version of the algorithm reported earlier by Chakraborty & Dill (1997)

predecessors of event $i$ have topological indices less than $i$. So, the entries from the $\Delta$ matrix that are used in the expressions for the initial estimates belong to layers numbered less than $i$. Since the entries in the matrix are computed in order of increasing layers, these entries would have already been computed by the time layer $i$ is processed. After the initial estimates are computed, they are *refined* in steps (2a) through (2d) of figure 5. Once again, observations 1 and 2 are used, but this time the type of event $j$ is used to determine the expressions for the bounds. Note that refining an entry in layer $i$ requires knowledge of values of other entries in the same layer. However, by considering each $j$ in increasing order from 0 through $i-1$, the required entries are estimated and refined before they are used.

Let $p$ denote the maximum number of predecessors of an event in the timing constraint graph. The complexity of the AcyclicApproxSep algorithm is easily seen to be $O(n^2.p)$. Since $p$ is usually bounded above by a small constant, the complexity is $O(n^2)$ for all practical purposes. In the worst case, $p$ is $O(n)$, so the complexity is $O(n^3)$.

We now prove the correctness of algorithm AcyclicApproxSep. Given an acyclic timing constraint graph $G$, let the maximum achievable time separation from event $i$ to event $j$ be $\Delta^*(i, j)$.

**Theorem 1.** *For every ordered pair of events* $(i, j)$, $\Delta^*(i, j)$ *is bounded above by* $\Delta(i, j)$.

*Proof.* We show inductively that at the end of every step of the algorithm, $\Delta^*(i, j) \leq \Delta(i, j)$. The steps of the algorithm are assumed to be numbered in the sequence in which they are executed. For notational convenience, the last step of initialization is referred to as step 0.
*Basis:* The claim is trivially true at the end of initialization.
*Hypothesis:* Let the claim be true at the end of step $k$, where $k \geq 0$.
*Induction:* Consider step $k+1$. At most one entry of the $\Delta$ matrix is modified in this step. Let this be $\Delta(i, j)$. The new value of $\Delta(i, j)$ is given by the expression on the right hand side of one of the assignments labelled (2a)–(2d) and (i)–(iv) in figure 5. By hypothesis, the $\Delta$ matrix entries used in these expressions are upper bounds of the corresponding maximum achievable time separations. Therefore, by observations 1 and 2, the newly computed value of $\Delta(i, j)$ is an upper bound of $\Delta^*(i, j)$.

It has been further shown by Chakraborty (1998) that the bounds computed by algorithm AcyclicApproxSep are *exact* if (i) the timing constraint graph has a single source event, and (ii) all events are of the same type (either all max or all min). We omit the proof here due to lack of space, and also because the result is not directly relevant to the algorithm developed in § 5. Although experiments reported in § 4.3 indicate that our algorithm computes exact bounds in several cases where the above conditions are not met, currently we do not have a better characterization of systems for which our algorithm produces exact results.

## 4.3 *Experiments with acyclic graphs*

The AcyclicApproxSep algorithm has been implemented and applied to several system specifications. The benchmarks include cyclic timing constraint graphs from Hulgaard *et al* (1994) and Myers & Meng (1993) unfolded to various degrees. These are named "h$r$-$n$" and "m$r$-$n$" in table 1, where different values of $r$ indicate different cyclic graphs, and $n$ denotes the degree of unfolding. The benchmark "mcmill" has been adapted from McMillan & Dill's (1992) paper. The "diffeq" benchmark was obtained from analysis of an asynchronous differential equation solver chip (explained in § 6). The graphs for "Sat-$n$" were obtained by generating random 3-SAT formulae in $n$ variables and then transforming them to timing constraint

**Table 1.** Computing *all pairs* of separations. Times shown are on a MIPS R5000, 150 MHz machine with 128 MB memory. Entries with "–" indicate that the algorithm did not terminate within 24 h. Entries with "N/A" indicate unavailability of the corresponding number.

| Example name | # Events ($n$) | # Min constr. | # Max constr. | Time (s) AcyclicApproxSep | Time (s)[a] | Accuracy |
|---|---|---|---|---|---|---|
| h1-64 | 321 | 0 | 316 | 0.49 | 1.48 | 1.0 |
| h1-256 | 1281 | 0 | 1276 | 8.65 | 26.31 | 1.0 |
| h1-512 | 2561 | 0 | 2556 | 36.69 | 113.77 | 1.0 |
| h2-64 | 385 | 0 | 379 | 0.75 | 2.21 | 1.0 |
| h2-256 | 1537 | 0 | 1531 | 13.17 | 40.31 | 1.0 |
| h2-512 | 3073 | 0 | 3067 | 55.63 | 171.86 | 1.0 |
| m1-64 | 641 | 0 | 255 | 1.55 | 4.97 | 1.0 |
| m1-256 | 2561 | 0 | 1023 | 27.87 | 92.05 | 1.0 |
| m1-512 | 5121 | 0 | 2047 | 120.69 | 417.89 | 1.0 |
| mcmill | 13 | 1 | 4 | <0.01 | 0.01 | 1.0 |
| diffeq | 288 | 4 | 42 | 0.21 | 12309.3 | 1.0 |
| Sat-3 | 20 | 4 | 9 | <0.01 | 0.9 | 0.997 |
| Sat-6 | 36 | 7 | 16 | <0.01 | 85.81 | 0.999 |
| Sat-8 | 44 | 9 | 18 | <0.01 | 287.42 | 1.0 |
| c1355 | 948 | 252 | 285 | 2.78 | – | N/A |
| c7552 | 5258 | 1174 | 1169 | 133.14 | – | N/A |
| c6288 | 4189 | 1672 | 1684 | 65.38 | – | N/A |
| c3540 | 3162 | 417 | 413 | 33.31 | – | N/A |

# Min constr.: Number of min constraints with 2 or more arguments
# Max constr.: Number of max constraints with 2 or more arguments
[a]Time using McMillan & Dill's (1992) algorithm

graphs using the technique outlined by McMillan & Dill (1992). The other benchmarks were obtained as systems of timing constraints during timing simulation of ISCAS85 circuits with randomly generated input vectors. For each benchmark, table 1 reports the number of events ($n$), and the number of min and max constraints with two or more arguments. To quantify the accuracy of the computed results, we have also computed the fraction of all $n^2$ time separations that were computed exactly by our algorithm. The exact time separation bounds for all the benchmarks were obtained using McMillan & Dill's algorithm (1992), whenever it terminated successfully within 24 h. Comparing the exact bounds with those computed by the proposed algorithm indicates that our results have a fairly high degree of accuracy in practice.

Table 1 also shows the time taken by the AcyclicApproxSep algorithm to analyse each benchmark on a MIPS R5000 processor running at 150 MHz with 128 MB of main memory. These times do not include the time required to read in the timing constraint graph and topologically sort the events, which are not significant. For max-only problems, the performances of the AcyclicApproxSep algorithm and McMillan & Dill's algorithm (1992) are comparable. The larger run-times of McMillan & Dill's (1992) algorithm are possibly attributable to a non-optimal implementation. However, for systems with both min and max timing constraints, the run times of the AcyclicApproxSep algorithm are signifi-

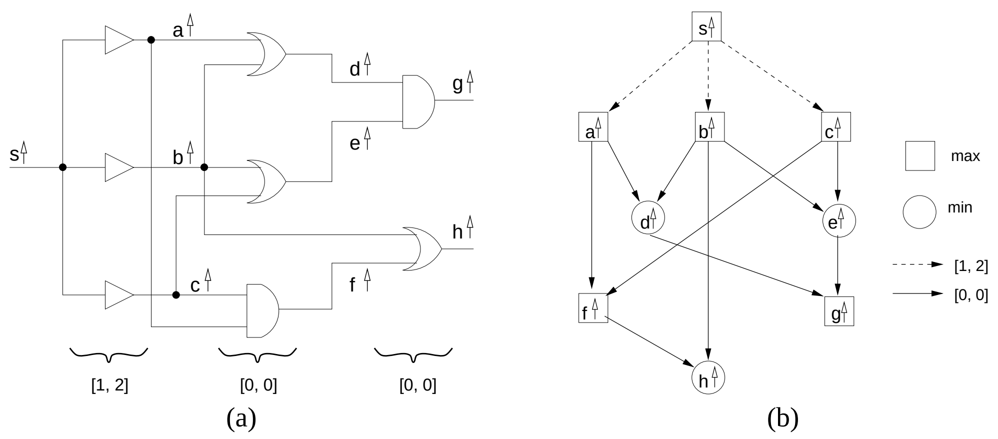


**Figure 6.** Pathological example: **(a)** Circuit and stimulus. Gate delays are [min, max]. Wires have zero delay. **(b)** Timing constraing graph.

cantly smaller than the those of the exact branch-and-bound technique of McMillan & Dill (1992).

## 4.4 *Pathological example*

Although the results in the previous subsection indicate that our algorithm generates results with a high degree of accuracy in practice, there are pathological cases as well. This is not surprising since the problem of computing exact time separation bounds is NP-complete, while our algorithm generates approximate solutions in polynomial time. In this section, we describe a pathological example for which the AcyclicApproxSep algorithm fails to compute all pairs of time separations exactly.

Consider the combinational circuit and input stimulus shown in figure 6a; where a $\uparrow$ indicates a rising transition. Gate delays are assumed to vary within the intervals shown, and wires are assumed to have zero delays. Figure 6b shows the timing constraint graph for this

**Table 2.** $\Delta$ matrix for pathological example.

| | s | a | b | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|---|---|
| s | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| a | -1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| b | -1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| c | -1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| d | -1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| e | -1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| f | -1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| g | -1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| h | -1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |

For clarity, the $\uparrow$s have been omitted from the row and column headings.

system – vertices represent signal transitions and edge delays represent delays of gates. The $\Delta$ matrix computed by applying the AcyclicApproxSep algorithm is shown in table 2.

It may be verified that all bounds except $\Delta(g, h)$ are exact. The computed value of $\Delta(g, h)$ is 1; however, the maximum achievable separation from the rising transition on $g$ to that on $h$ is 0. To see why the algorithm fails to compute the correct bound, observe that the rising transition on $h$ is a min type event whereas that on $g$ is a max type event. Therefore, the value of $\Delta(g, h)$, computed by the AcyclicApproxSep algorithm, is given by $\min(\Delta(g, f) + D_{f,h}, \Delta(g, b) + D_{b,h})$. In order for this to represent the maximum achievable time separation from $g$ rising to $h$ rising, one of the following conditions must be satisfied.

- $f$ rises $\Delta(g, f)$ $(= 1)$ time units after $g$, and the rising transition on $f$ causes $h$ to rise after $D_{f,h}$ $(= 0)$ time units,
- $b$ rises $\Delta(g, b)$ $(= 1)$ time units after $g$, and the rising transition on $b$ causes $h$ to rise after $D_{b,h}$ $(= 0)$ time units.

Unfortunately, it can be seen from the dependencies in the timing constraint graph, that if $f$ rises 1 time unit after $g$, then $b$ must rise 1 time unit before $f$. Since the rising transition on $h$ is a min-type event, the transition on $b$, not that on $f$, causes $h$ to rise. Similarly, if $b$ rises 1 time unit after $g$, $f$ must rise 1 time unit before $b$, and therefore, the transition on $b$, not that on $f$, causes $h$ to rise. The min expression used by the algorithm to compute the value of $\Delta(g, h)$ fails to take into account this complex dependency between the times of occurrence of the transitions on $f$, $b$, $g$ and $h$. Consequently, the value given by the min expression is a conservative upper bound of the corresponding maximum achievable separation.

## 5. Analysing tightly-coupled systems

We now use algorithm AcyclicApproxSep to develop an efficient technique for computing time separation bounds in tightly-coupled systems. Recall from § 2.2 that subgraph $G_i$ of the infinite unfolded graph is isomorphic to $G_1$ for all $i \geq 1$. It therefore suffices to analyse subgraphs $G_0$ and $G_1$ to compute the bounds of interest in tightly-coupled systems.

Our approach consists of two phases, as follows.

- The first phase uses a successive refinement approach to compute bounds of the time separation of events after the system has run for a sufficiently long period of time. Specifically, if $K$ denotes the number of successive refinement iterations, the computed bounds apply to every subgraph $G_i$ for $i \geq K$. This is explained in § 5.1. Since the bounds computed in this phase apply only after the system has run for some time, these are called *long-term time separation bounds* and this phase is called *long-term behaviour analysis*.
- Assuming that the algorithm iterates $K$ times in the first phase, the second phase computes time separations between events in the finite acyclic graph modelling the behavior of the system from Reset to the events in cutset $C_K$.

Bounds of the time separation of events in $G_0$ are obtained directly from the second phase. To obtain an upper bound of the time separation of a pair of events in $G_i$, maximised over all $i \geq 1$, the maximum of the corresponding bounds computed for subgraphs $G_1$ through $G_K$ is determined. The bounds for $G_1$ through $G_{K-1}$ are obtained from the second phase, and the bounds for $G_K$ (and for all $G_i$ with $i > K$) are obtained from the first phase. The following subsections describe each phase of the algorithm in greater detail.

## 5.1 *Phase I of analysis*

We first concentrate on computing bounds of the time separation of events in subgraph $G_1$. The isomorphism between $G_1$ and $G_i$ for all $i \geq 1$ is then exploited to compute bounds of the time separation of events in subgraphs $G_i$ for $i \geq 1$.

Given an event $v$ in $G_1$, let $\Pi(v)$ denote the set of paths from events in cutset $C_1$ to $v$. The *shortest* and *longest path lengths* to $v$, denoted $l(v)$ and $L(v)$, are defined as follows: $l(v) = \min_{P \in \Pi(v)} \left( \sum_{\langle x,y \rangle \text{ along } P} d_{x,y} \right)$, and $L(v) = \max_{P \in \Pi(v)} \left( \sum_{\langle x,y \rangle \text{ along } P} D_{x,y} \right)$. Note that if $v$ is in $C_1$, there exists a degenerate path $(v)$ from $v$ to itself, so $l(v)$ is at most 0 and $L(v)$ is at least 0. Since edge delays in a timing constraint graph model component delays in a system or its environment, edge delay bounds may be finite or infinite in general. An infinite delay bound may be required, for example, to model the response time of an environment whose delay characteristics are unknown. In this paper, however, all systems and environments are assumed to have finite, non-negative delay bounds. Since subgraph $G_1$ is finite, it follows that $l(v)$ and $L(v)$ are non-negative and finite for all events $v$ in $G_1$. A brief discussion of the implications of infinite edge delay bounds is given in § 7.

Since $G_1$ lies between cutsets $C_1$ and $C_2$, the source events of $G_1$ are elements of cutset $C_1$. Therefore, in order to apply algorithm AcyclicApproxSep to $G_1$, the values of $\Delta(a_1, b_1)$ for every pair of events, $a_1$ and $b_1$, in $C_1$ are required. Normally, this may be obtained by an analysis of subgraph $G_0$. However, we choose to decouple the analysis of the initial behaviour from the long-term behaviour analysis and assume that the initial behaviour is unknown during phase I of the analysis. Thus, the entries $\Delta(a_1, b_1)$ for every $a_1$ and $b_1$ in $C_1$ are conservatively set to $+\infty$ and algorithm AcyclicApproxSep is invoked. As will be seen later, this ensures that the long-term behaviour analysis terminates within a pseudo-polynomial number of steps, while producing finite, conservative bounds of the long-term time separation of events. Since no assumptions are made about the initial behaviour of the system, the time separation bounds computed in this phase apply regardless of the actual initial behaviour. Therefore, an upper bound computed in this phase can be viewed as maximised over all possible initial behaviours.

In general, setting the upper bound of the time separation between every pair of source events to $+\infty$ can lead to the computed bounds between other pairs of events being $+\infty$ as well. However, we have the following result for tightly-coupled systems.

**Theorem 2.** *For every pair of events $u$ and $v$ in $C_2$, $\Delta(u, v)$ computed by algorithm* Acyclic-ApproxSep *is bounded above by $L(v) - l(u)$.*

*Proof.* Let there be $n$ events in graph $G_1$. Let $|C_1| = |C_2| = k$. Without loss of generality, we assign indices 0 through $k-1$ to events in $C_1$, and $n-k$ through $n-1$ to those in $C_2$. The theorem is proved by establishing a stronger result: $\Delta(i, j) \leq L(j) - l(i)$ for $n-k \leq i \leq n-1$, and $0 \leq j \leq n-1$.

Let $n-k \leq i \leq n-1$. Our proof consists of three parts, corresponding to three partitions of the range $[0, n-1]$.

*Part* I. Let $0 \leq j \leq k-1$. Since $i > j$, $\Delta(i, j)$ is computed in the $i$th iteration of the outermost loop of algorithm AcyclicApproxSep (see figure 5).

By property P4 in definition 1, there exists a path $P$ from event $j$ to event $i$ such that all events along $P$ are of max type. Let $v$ be the predecessor of $i$ along $P$. It follows from step (i) of function ComputeInitialEstimates (see figure 5) that

$$\Delta(i, j) \leq \Delta(v, j) - d_{v,i}. \tag{9}$$

Since event $v$ lies on path $P$, it must be of max type by property P4. In addition, if event $j$ is not the same as event $v$, we must have $j < v$ since $P$ is a path from event $j$ through event $v$ to event $i$. Let $u$ be the predecessor of $v$ along $P$. From step (i) of function ComputeInitialEstimates, we again have

$$\Delta(v, j) \leq \Delta(u, j) - d_{u,v}. \tag{10}$$

Continuing this argument and adding the resulting inequalities, we obtain

$$\begin{aligned} \Delta(i, j) &\leq \Delta(j, j) \quad - \sum_{\langle x,y\rangle \text{ along } P} d_{x,y} \\ &\leq -l(i) \qquad (11) \\ &\leq L(j) - l(i). \qquad (12) \end{aligned}$$

Inequality (11) follows from the observation that $\Delta(j, j)$ is 0 by definition and $l(i)$ is at most $\sum_{\langle x,y\rangle \text{ along } P} d_{x,y}$. Inequality (12) follows from the observation that $j$ is in $C_1$, so $L(j)$ is at least 0 by definition.
*Part* II. Let $k \leq j \leq i - 1$. Once again, $i > j$ and $\Delta(i, j)$ is computed in the $i$th iteration of the outermost loop of algorithm AcyclicApproxSep.
The proof is by induction on $j$.
*Basis*. From part I, $\Delta(i, j) \leq L(j) - l(i)$ for $0 \leq j \leq k - 1$.
*Hypothesis*. Let $k \leq j \leq i - 1$. For all $r$ in 0 through $j - 1$, let $\Delta(i, r) \leq L(r) - l(i)$.
*Induction*. It follows from steps (2a) and (2c) of algorithm AcyclicApproxSep that

$$\begin{aligned} \Delta(i, j) &\leq \max_{k \,\in\, preds(j)} (\Delta(i, k) + D_{k,j}) \\ &\leq \max_{k \,\in\, preds(j)} (L(k) - l(i) + D_{k,j}) \qquad (13) \\ &\leq L(j) - l(i). \qquad (14) \end{aligned}$$

Inequality (13) follows from the hypothesis, and inequality (14) follows from the observation that the longest path length to $j$ is no smaller than the longest path length to any predecessor of $j$ plus the maximum delay of the corresponding edge to $j$.
*Part* III. Finally, consider $i + 1 \leq j \leq n - 1$. The proof is by induction on $j$.
*Basis*. Follows from parts (I) and (II).
*Hypothesis*. Let $i + 1 \leq j \leq n - 1$. For all $r$ in 0 through $j - 1$, let $\Delta(i, r) \leq L(r) - l(i)$.
*Induction*. Since $i < j$, $\Delta(i, j)$ is computed in the $j$th iteration of the outermost loop of algorithm AcyclicApproxSep. It follows from steps (ii) and (iv) of function ComputeInitialEstimates that

$$\Delta(i, j) \leq \max_{k \,\in\, preds(j)} (\Delta(i, k) + D_{k,j}). \tag{15}$$

Using an argument similar to that used in the induction step of part (II), the value of $\Delta(i, j)$ is seen to be bounded above by $L(j) - l(i)$.

COROLLARY 1

*For every pair of events $u$ and $v$ in $C_2$, $\Delta(u, v)$ computed by algorithm* AcyclicApproxSep *is finite.*

*Proof.* Follows from theorem 2 and the observation that $\Delta(i, j) \neq -\infty$. The latter is true because the $\Delta$ matrix is initialized with $+\infty$ and other finite values, and then finite edge delay bounds are added or subtracted finitely many times.

The repetitive structure of the unfolded graph now suggests the following iterative procedure for computing bounds of the time separation of events in subgraphs $G_i$ for $i \geq 1$. Although the algorithm actually analyses subgraph $G_1$ in each iteration, successive iterations may be viewed as analysis of subgraphs $G_1$, $G_2$, $G_3$ and so on, each of which is isomorphic to $G_1$. It is assumed that the bounds computed in iteration $i$ are stored in an $n \times n$ matrix, $\Delta_i$. The first iteration starts by setting $\Delta_1(a_1, b_1)$ to $+\infty$, for every pair of events $a_1$ and $b_1$ in $C_1$. Algorithm AcyclicApproxSep is then invoked and bounds of the time separations of all pairs of events in $G_1$ are obtained. By corollary 1, the computed bounds between events in $C_2$ are finite. Let $a_2$ and $b_2$ be events in $C_2$ and $a_1$ and $b_1$ be the corresponding events in $C_1$. The value of $\Delta_1(a_2, b_2)$ is now copied to $\Delta_2(a_1, b_1)$ for every pair of events $a_2$ and $b_2$ in $C_2$, and algorithm AcyclicApproxSep re-invoked to compute bounds of the time separations of all pairs of events in $G_2$. The intuition behind this operation is that the second iteration corresponds to analysis of subgraph $G_2$, and the source events in $G_2$ are elements of cutset $C_2$. In general, this process can be repeated: at the end of iteration $i$, the value of $\Delta_i(a_2, b_2)$ is copied to $\Delta_{i+1}(a_1, b_1)$ for every pair of events $a_2$ and $b_2$ in $C_2$, as input conditions for iteration $i + 1$. Bounds of the time separations of all pairs of events in $G_{i+1}$ are then computed in iteration $i + 1$ by invoking algorithm AcyclicApproxSep.

The iterative analysis may be terminated when one of the following conditions is satisfied.

(a) $\Delta_i(a_1, b_1)$ equals $\Delta_i(a_2, b_2)$, for all pairs of events, $a_1$ and $b_1$, in $C_1$. The analysis is said to have *converged* in this case.
(b) The number of iterations reaches a user-specified upper bound, $K_{max}$.

If all edge delay bounds are integers, an upper bound on the number of iterations required to converge can be derived (see lemma 2 below). However, in practice, the analysis usually converges within a small number of iterations – at most 5 in our experiments. We conjecture that by choosing $K_{max}$ to lie in the range 10 to 20, the analysis can be expected to converge in most practical systems. Condition (b) ensures that the analysis terminates after a pre-set number of iterations if it has not already converged by then.

The pseudo-code in phase I of figure 7 describes the iterative algorithm outlined above. If there are $n$ events in $G_1$ and $p$ denotes the maximum number of predecessors of an event, the complexity of algorithm AcyclicApproxSep, as noted in § 4.2, is $O(n^2.p)$. The complexity of phase I is therefore $O(K_{max}.n^2.p)$.

In order to characterize the bounds computed in phase I, the $\leq_{\mathbf{M}}$ relation for $\Delta$ matrices is defined as follows. Let $\Delta$ and $\Delta'$ be two $n \times n$ matrices. $\Delta$ is said to be bounded above by $\Delta'$, denoted $\Delta \leq_{\mathbf{M}} \Delta'$, if for all $u$ and $v$ in 0 through $n - 1$, the value of $\Delta(u, v)$ is no larger than the value of $\Delta'(u, v)$.

*Lemma* 1. *For all iteration indices i in phase I,* $\Delta_{i+1} \leq_{\mathbf{M}} \Delta_i$. *This property is called* ***monotonic convergence*** *of the time separation intervals.*

*Proof.* We start with the following observation about algorithm AcyclicApproxSep.

*Observation* 3. After initialization, the only steps in which the $\Delta$ matrix is updated are steps (i) through (iv) in ComputeInitialEstimates and steps (2a) through (2d) in AcyclicApproxSep. The expressions on the right hand side of assignments in each of these steps involve non-negated

```
CyclicApproxSep ($C_1$, $C_2$: cutsets; $G_0$, $G_1$: acyclic subgraphs; $K_{max}$: integer )
  Phase I:
  1. /* Initialization */
     (a) $i = 1$;
     (b) for all ordered pairs $(a_1, b_1)$ in $C_1$
             $\Delta_1(a_1, b_1) = +\infty$;
  2. do
     (a) AcyclicApproxSep ($G_1$, $\Delta_i$);
     (b) if $((\Delta_i(a_1, b_1) == \Delta_i(a_2, b_2)$ for every ordered pair $(a_1, b_1)$ in $C_1)$ OR
               $(i == K_{max}))$
           (i) $K = i$;   /* no. of iterations so far */
           (ii) go to Phase II;
         else
           (iii) for all ordered pairs $(a_1, b_1)$ in $C_1$
                   $\Delta_{i+1}(a_1, b_1) = \Delta_i(a_2, b_2)$;
           (iv) $i = i + 1$;
  Phase II:
  1. Construct $G_{startup}$, the acyclic graph composed of $G_0$ followed by
     $K - 1$ instances of $G_1$.
     /* Note that $G_{startup}$ has only one source event */
  2. AcyclicApproxSep ($G_{startup}$, $\Delta$);
```

**Figure 7.** Timing analysis algorithm for tightly-coupled systems.

$\Delta$ matrix entries. Therefore, if all edge delay bounds are left unchanged and bounds between every pair of source events are either decreased or left unchanged, the bounds computed by algorithm AcyclicApproxSep cannot increase.

The lemma is now proved by induction on $i$.
*Basis* $(i = 1)$. In the first iteration, $\Delta_1(a_1, b_1) = +\infty$ for every pair of events $a_1$ and $b_1$ in $C_1$. In the second iteration, $\Delta_2(a_1, b_1) = \Delta_1(a_2, b_2)$, where $a_2$ and $b_2$ are the corresponding events in $C_2$. However, by corollary 1, $\Delta_1(a_2, b_2)$ is finite, so $\Delta_2(a_1, b_1)$ is finite. It follows from observation 3 that $\Delta_2 \leq_{\mathbf{M}} \Delta_1$.
*Hypothesis*. For $i \geq 1$, let $\Delta_{i+1} \leq_{\mathbf{M}} \Delta_i$.
*Induction*. By hypothesis, $\Delta_{i+1}(a_2, b_2) \leq \Delta_i(a_2, b_2)$ for every pair of events $a_2$ and $b_2$ in $C_2$. However, by step 2(b)(iii) of phase I of algorithm CyclicApproxSep (see figure 7), $\Delta_{i+1}(a_2, b_2)$ equals $\Delta_{i+2}(a_1, b_1)$ and $\Delta_i(a_2, b_2)$ equals $\Delta_{i+1}(a_1, b_1)$. Therefore, $\Delta_{i+2}(a_1, b_1) \leq \Delta_{i+1}(a_1, b_1)$, for every pair of events $a_1$ and $b_1$ in $C_1$. The proof then follows from observation 3.

*Lemma* 2. *For event $v$ in subgraph $G_1$, let $l(v)$ and $L(v)$ be the shortest and longest path lengths from events in $C_1$ to $v$. If cutset $C_2$ has $r$ events, and all edge delay bounds are integers, Phase I converges in at most* $(2 + (r - 1) \cdot \sum_{v \in C_2}(L(v) - l(v)))$ *iterations.*

*Proof.* The constant 2 accounts for the first iteration and the final iteration in which convergence is detected. The lemma is proved by showing that there can be at most $(r - 1) \cdot \sum_{v \in C_2}(L(v) - l(v))$ iterations in between.

First, note that for every pair of events $a$ and $b$ in $G_1$, $-\Delta(b, a) \leq \tau_b - \tau_a \leq \Delta(a, b)$. Therefore, $\Delta(a, b) \geq -\Delta(b, a)$.

From theorem 2, we know that for every pair of distinct events $u$ and $v$ in $C_2$, $\Delta(u, v) \leq L(v) - l(u)$ after the first iteration. Lemma 1 implies that in every subsequent iteration, the value of $\Delta(u, v)$ either remains unchanged or decreases. The minimum value to which it can decrease is $-\Delta(v, u)$, which is bounded below by $l(v) - L(u)$. Therefore, the range of variation of $\Delta(u, v)$ is $(L(v) - l(v)) + (L(u) - l(u))$. The same range applies to the variation of $\Delta(v, u)$ as well.

Since all edge delay bounds are integral, $\Delta(u, v)$ and $\Delta(v, u)$ can decrease only by integral amounts. Therefore, after the first iteration, there can be at most $(L(v) - l(v)) + (L(u) - l(u))$ iterations in which the values of $\Delta(u, v)$ and $\Delta(v, u)$ decrease. Summing this over all pairs of distinct events in $C_2$ gives $(r - 1) \cdot \sum_{v \in C_2} (L(v) - l(v))$ as the maximum number of iterations after the first iteration during which any entry $\Delta(u, v)$, with $u$ and $v$ in $C_2$, can decrease. Since the algorithm converges when none of the entries $\Delta(u, v)$ decrease any further, this proves the lemma.

**Theorem 3.** *Let phase I terminate after $K$ iterations, with $K$ at least 2. The bounds computed in iteration $K$ are finite and apply to events in every subgraph $G_i$ for $i \geq K$, regardless of the initial behaviour of the system.*

*Proof.* If at least two iterations are made, corollary 1 and step 2(b)(iii) of phase I of algorithm CyclicApproxSep (see figure 7) imply that the bounds between every pair of source events in $C_1$ are finite. Since all edge delay bounds are also finite, it is easy to see from the steps in algorithm AcyclicApproxSep that all $\Delta$ matrix entries computed by the algorithm are finite.

The remainder of the theorem is proved by showing that if the analysis is allowed to continue for $i$ iterations where $i > K$, then the upper bounds of time separations computed in iteration $i$ apply to the events in subgraph $G_i$. Since $\Delta_i \leq_{\mathbf{M}} \Delta_K$ for all $i \geq K$ (by lemma 1), the theorem follows.

*Basis* ($i = 1$). The first iteration is viewed as analysis of $G_1$. Note that the source events of $G_1$ belong to cutset $C_1$. Since $\Delta_1(a_1, b_1)$ is set to $+\infty$ for every $a_1$ and $b_1$ in $C_1$, the value of $\Delta_1(a_1, b_1)$ is indeed an upper bound of the actual time separation from event $a_1$ to event $b_1$. Therefore, by theorem 1, $\Delta(u_1, v_1)$ computed by algorithm AcyclicApproxSep is an upper bound of the exact maximum separation from $u_1$ to $v_1$, for every pair of events $u_1$ and $v_1$ in $G_1$. Similarly, the separation from $a_2$ to $b_2$ is bounded above by $\Delta_1(a_2, b_2)$ for every pair of events $a_2$ and $b_2$ in cutset $C_2$.

*Hypothesis*. Let the bounds computed in iteration $i - 1$ apply to all events in $G_{i-1}$, for $i \geq 2$.

*Induction*. The $i$th iteration is viewed as analysis of $G_i$. The source events of $G_i$ belong to cutset $C_i$. By hypothesis, the separation from $a_i$ to $b_i$, for every pair of events $a_i$ and $b_i$ in $C_i$, is bounded above by $\Delta_{i-1}(a_2, b_2)$. Since the value of $\Delta_{i-1}(a_2, b_2)$ is copied to $\Delta_i(a_1, b_1)$ at the end of iteration $i - 1$, the value of $\Delta_i(a_1, b_1)$ is an upper bound of the time separation from $a_i$ to $b_i$. The proof then follows from the fact that algorithm AcyclicApproxSep computes conservative upper bounds of time separation of events.

## 5.2 *Phase II of analysis*

Suppose phase I terminates after $K$ iterations. By theorem 3, the bounds in $\Delta_K$ apply to events in every subgraph $G_i$ for $i \geq K$. To compute bounds of the time separation of events in $G_0$ through $G_{K-1}$, the initial segment of the unfolded graph, consisting of subgraphs $G_0$ through $G_{K-1}$ is constructed and analysed using the AcyclicApproxSep algorithm. This is shown in phase II of the pseudo-code in figure 7.

If subgraph $G_0$ has $m$ events and subgraph $G_1$ has $n$ events, constructing the acyclic subgraph comprised of $G_0$ through $G_{K-1}$ requires at most $O(m^2 + K_{max}.n^2)$ time. The resulting graph has at most $m + K_{max}.n$ events. If $p$ denotes the maximum number of predecessors of an event, algorithm AcyclicApproxSep takes $O((m + K_{max}.n)^2.p)$ time to analyse this graph. Therefore, the complexity of phase II is $O((m + K_{max}.n)^2.p)$. To compute bounds of the time separation of a pair of events in $G_i$, maximised over all $i \geq 1$, at most $K_{max}$ additional comparisons are needed to determine the maximum of the corresponding bounds computed for $G_0$ through $G_K$. Repeating this for every pair of events in $G_i$ requires $O(K_{max}.n^2)$ comparisons. Therefore, the complexity of the entire procedure is $O((m + K_{max}.n)^2.p)$.

There are three sources of conservativeness in the above analysis.

(1) Since we set the time separation of every ordered pair of events in $C_1$ to $+\infty$ at the beginning of phase I, the computed long-term time separation bounds may be conservative. This can happen if, for example, the exact long-term bounds depend on the initial behaviour of the system. The reader may wonder why we need to start the analysis assuming no knowledge of the initial behaviour. However, as may be seen from the basis case of the proof of lemma 1, this guarantees monotonic convergence of the bounds in phase I, which, in turn, guarantees that the analysis converges within a finite number of steps (see proof of lemma 2).
(2) Algorithm AcyclicApproxSep may return conservative bounds in the worst-case.
(3) If we terminate phase I by hitting $K_{max}$, rather than by converging, the computed bounds may be conservative.

In our experiments, however, all the computed bounds were found to be exact.

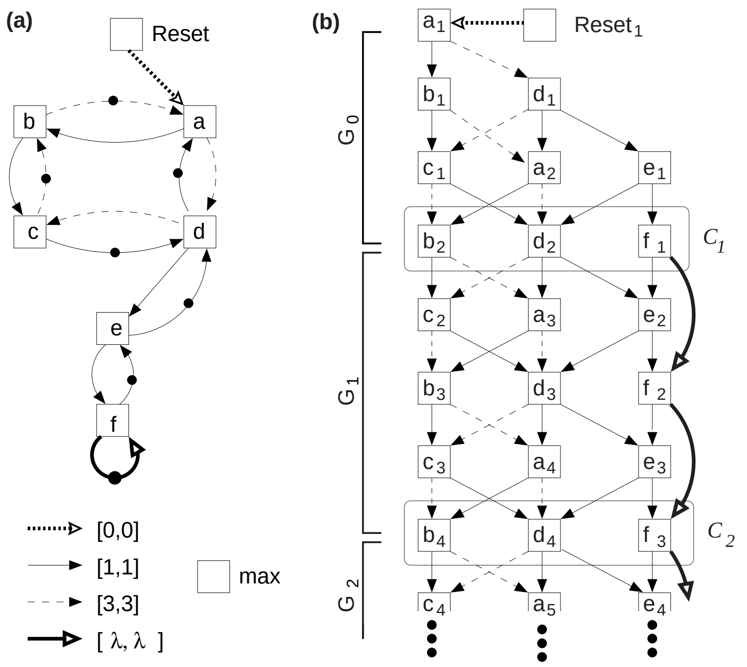

(a)
Reset
a
b
c
d
e
f
[0,0]
[1,1]
[3,3]
[ λ, λ ]
max
(b)
Reset1
a1 b1 d1 c1 a2 e1 b2 d2 f1 c2 a3 e2 b3 d3 f2 c3 a4 e3 b4 d4 f3 c4 a5 e4
G0
G1
G2
C1
C2

**Figure 8.** Analysing the example by Amon (1993). Edge delays are as in the legend. **(a)** Cyclic timing constraint graph; **(b)** initial part of unfolded graph.

**Table 3.** Time separation bounds for system of figure 8. $M^i$ denotes the exact upper bound of $\tau_{a_i} - \tau_{a_{i-1}}$ for $i \geq 2$ .

| If $\lambda = 6$: | | If $\lambda = 9$: | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $M^{\text{even}}$ | $M^{\text{odd}}$ | $M^2$ | $M^3$ | $M^4$ | $M^5$ | $M^6$ | $M^7$ | $M^8$ | $M^9$ | $M^{\geq 10}$ |
| 4 | 8 | 4 | 8 | 4 | 8 | 4 | 8 | 8 | 9 | 9 |

### 5.3 *An example*

*Example* 2. The example, shown in figure 8a, is adapted from the work by Amon (1993). This example demonstrates that the number of iterations required to converge depends on the edge delay bounds in general.

Let $M^i$ denote the exact upper bound of $\tau_{a_i} - \tau_{a_{i-1}}$, maximised over all $i$, for $i$ greater than or equal to two.[2] It has been shown by Amon (1993) that $M^i$ depends on both $i$ and $\lambda$, as indicated in table 3.

Figure 8b shows an initial fragment of the unfolded graph. For clarity, the events in $G_1$ have not been relabelled to have occurrence index 1 – the relabelling was simply an artifact for maintaining consistent notation when proving properties of the algorithm. The time separation of interest is that from $a_3$ to $a_4$.

Applying algorithm CyclicApproxSep, we find that phase I converges after 2 iterations when $\lambda$ is 6, and the computed value of $\Delta_2(a_3, a_4)$ equals 8. With $\lambda = 9$, phase I converges after 5 iterations and $\Delta_5(a_3, a_4)$ equals 9. This demonstrates that the number of iterations depends on the edge delay bounds in general.

From the above results and theorem 3, we conclude that if $\lambda$ equals 6, $M^i$ is bounded above by 8 for all even $i$ greater than or equal to 4. Similarly, if $\lambda$ equals 9, then $M^i$ is bounded above by 9. Now, it is easy to see that if the cutsets are shifted by one occurrence index, so that $C_1$ is $\{b_3, d_3, f_2\}$ and $C_2$ is $\{b_5, d_5, f_4\}$, the same analysis applies and $\Delta_2(a_4, a_5)$ equals 8 when $\lambda$ is 6, and $\Delta_5(a_4, a_5)$ equals 9 when $\lambda$ is 9. Therefore, $M^i$ is bounded above by these numbers for all odd $i$ greater than or equal to 5. These match the exact bounds computed by Amon (1993).

## 6. Analysing an asynchronous chip

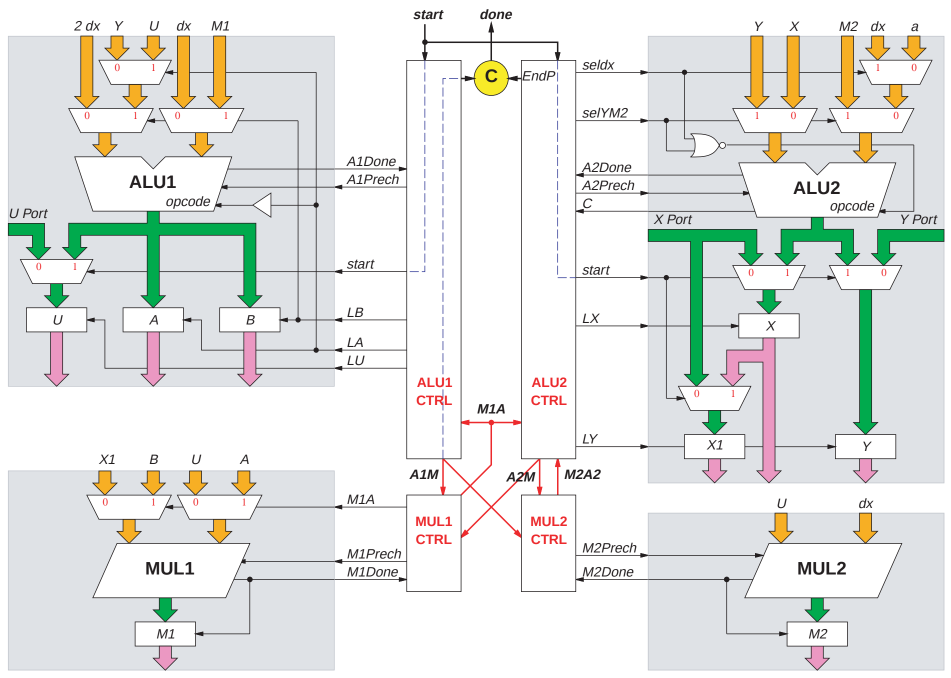
We now apply the CyclicApproxSep algorithm to verify timing constraints and to estimate performance bounds of an asynchronous differential equation solver chip. A known timing bug in a preliminary version of the design, that could be detected only after several hours of SPICE simulation, was uncovered in a few seconds by the analysis.

### 6.1 *Chip overview*

Figure 9, adapted from Yun *et al* (1998), shows the architecture of the chip and the dataflow graph for one iteration of operation. The system iterates through the operations depicted in the dataflow graph until a termination condition is satisfied. In this paper,

[2] $M^i$ in this notation equals $\Delta^{i-1}$ in the notation of Amon (1993)

(b)



(a)

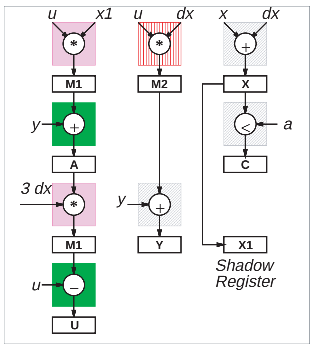


**Figure 9.** **(a)** Dataflow graph; **(b)** Architecture of Yun *et al*’s (1998) differential equation solver.

however, we are interested only in the behaviour of the system as it iterates through the dataflow graph, so the termination condition is ignored. This, of course, implies that the computed results do not apply to the final iteration when the system chooses to terminate the computation. The behaviour in the final iteration must be analysed separately, and is not addressed in this paper. Details of the DiffEq design are described by Yun *et al* (1998).

The control logic of DiffEq is implemented using four distributed *extended burst-mode* (XBM) controllers (Yun 1994): ALU1Ctrl, ALU2Ctrl, MUL1Ctrl and MUL2Ctrl. The controllers communicate with one another using a handshaking protocol that implicitly assumes safe timing bounds in signaling. Like synchronous systems, the control is responsible for generating ALU opcodes, multiplexer selects, register load enables, and precharge/evaluate signals. However, there are implicit assumptions on the timing of datapath signals, e.g., data inputs to domino circuits must stabilize before evaluation begins.

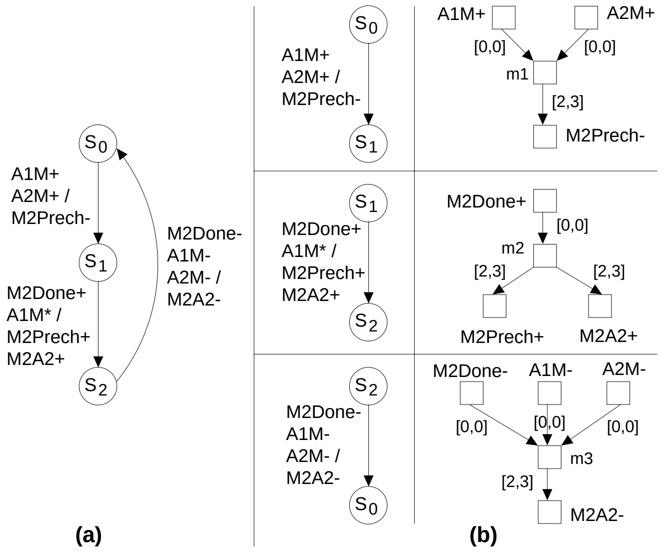


**Figure 10.** Modelling temporal behaviour of MUL2 CTRL. **(a)** MUL2 controller; **(b)** timing constraint graph fragments (controller delay = [2, 3]).

## 6.2 *Modelling controller timing*

We view each controller as a “black box” that receives input stimuli from its environment and responds by asserting values on its output ports after some internal delay. This model suffices for performance analysis, and datapath and inter-controller protocol timing verification. Other timing constraints required for correct operation of individual XBM controllers may be obtained by the technique described by Chakraborty *et al* (1997) and are assumed to be satisfied.

Without the check for the termination condition, each XBM controller in DiffEq cycles through a deterministic sequence of states. To model a controller, we examine its state transition diagram. For each state transition $S_i \rightarrow S_j$, let $\mathcal{I}$ be the set of input transitions and $\mathcal{O}$ the set of enabled output transitions. XBM semantics require that all *terminating* transitions (Yun 1994) in $\mathcal{I}$ must occur before the state transition is enabled, and all transitions in $\mathcal{O}$ are concurrently enabled once all terminating transitions in $\mathcal{I}$ have occurred. To model this behaviour, we create a max type event, say $m$, and draw zero-delay edges from the events representing the terminating transitions in $\mathcal{I}$ to $m$. We also draw edges from $m$ to each event representing the concurrently enabled transitions in $\mathcal{O}$, and label these edges with the delay incurred by the controller in generating the corresponding output transition after the last terminating transition in $\mathcal{I}$ has occurred.

As an example, figure 10a shows the XBM state transition diagram of MUL2Ctrl in DiffEq. A signal name with a $+$ indicates a rising transition, a name with a $-$ indicates a falling transition, while one with a $*$ denotes a *directed don’t care*. Details of the semantics of these transition types are described in Yun’s dissertation (Yun 1994). Figure 10b shows how each state transition is modelled using timing constraint graphs. Note that A1M is a directed don’t care during the state transition $S_1 \rightarrow S_2$. By XBM semantics, the controller need not wait for a transition on A1M before changing state from $S_1$ to $S_2$. This is reflected in figure 10b by the absence of an edge from a transition on A1M to $m_2$. However, A1M has a terminating falling transition during the next state transition $S_2 \rightarrow S_0$. So the controller must wait for A1M to fall before changing state from $S_2$ to $S_0$. This is represented by drawing an edge from A1M− to $m3$ in figure 10b.

The procedure outlined above generates timing constraint graph fragments for individual state transitions of XBM machines. Different fragments from the same machine as well as fragments from different machines are now "stitched" together by connecting input events in one fragment to output events in other fragments. For example, consider the following sequence of operations: MUL2Ctrl raises M2prech and in response, the MUL2 unit lowers M2done. If the delay from M2prech going high to M2done going low, plus the signal propagation delay from MUL2Ctrl to MUL2 and back lies between 4 and 5 time units, an edge is drawn from M2prech+ to M2done− and labelled [4,5]. Continuing this process, the timing constraint graph fragments of the different controllers can be stitched together to obtain a single graph modelling the behaviour of the four interacting controllers for one iteration of operation of DiffEq.

The above discussion concentrated on modelling choice-free XBM controllers. In general, timing constraint graphs can also be used to model other choice-free controllers if the temporal dependencies between all events can be expressed using min and max type constraints. Each input and output signal transition is represented by distinct events and if the output transition occurs only after all the inputs have transitioned, a max type constraint is used to connect them. This is the case, for example, in event-rule systems (Burns 1991), signal graphs (Nielsen & Kishinevsky 1994), choice-free XBM controllers etc. If the output transitions as soon as one of the inputs has transitioned (e.g., in some extended event-rule systems (Lee 1995)), a min type constraint is used instead.

### 6.3 *Modelling datapath timing*

This section describes a modelling technique for one important datapath component. The behaviour of other datapath components can be modelled similarly (see for example Chakraborty *et al* 1998, 1999).

6.3a *Arithmetic circuits designed in domino logic:* Figure 11a shows the ALU2 unit of DiffEq implemented in domino logic. The A2Done signal is used to notify other units that all internal nodes in ALU2 have been precharged (A2Done−) or that evaluation has completed (A2Done+). The value of the opcode input determines the type of operation (addition or subtraction) to be performed. When the precharge signal is asserted (A2Prech+), internal nodes of the circuit are pulled up to $V_{DD}$ after some precharging delay. This drives the outputs of the domino circuit low and also drives A2Done low. This behaviour is modelled by the directed edge from A2Prech+ to A2Done− in figure 11b.

The signal opcode at the output of the NOR gate in figure 11a falls as soon as one of its inputs rises. Therefore, the falling of opcode is modelled by a min type event (circle in figure 11b). This demonstrates the need for min type constraints when modelling datapath components at a low level of abstraction. Events Op1Stable and Op2Stable represent the stabilization of data operands, whereas InputsStable represents the stabilization of all inputs (both data operands and opcode). Evaluation begins as soon as A2Prech is de-asserted (A2Prech−). Assuming that InputsStable occurs before A2Prech− (this is checked during timing verification), the circuit computes the sum/difference of the operands and raises A2Done after some data-dependent computation delay. This is modelled as shown in figure 11b. The event InputsChanged in figure 11b represents a change in the value of one of the inputs after evaluation.

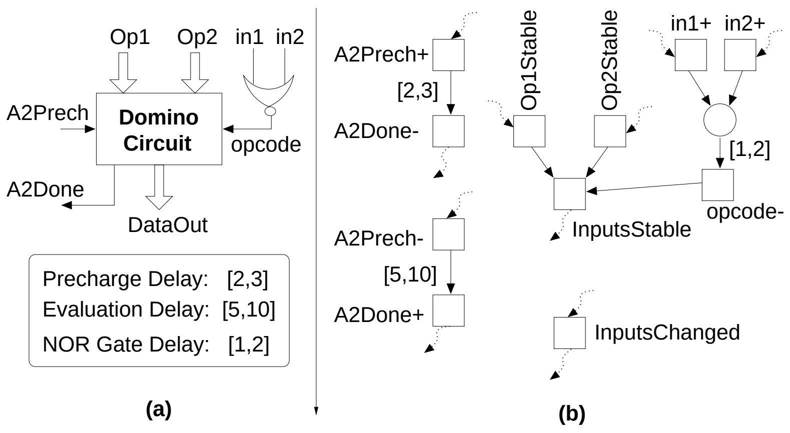


**Figure 11.** Modelling a domino circuit. **(a)** Circuit; **(b)** timing constraint graph: dotted edges represent connections to/from other parts of timing constraint graph. Unlabelled edges imply $[0, 0]$. Squares represent max-type events, circles represent min-type events.

## 6.4 *Formulating performance metrics and timing constraints*

In this section, performance metrics and timing constraints for correct operation of DiffEq are formulated as time separations between events. The objective is to apply the CyclicApproxSep algorithm to verify all timing constraints and obtain bounds on performance metrics of the system.

6.4a *Performance metrics:* We concentrate on two performance metrics.

- *Loop delay* – The delay from the start of the first operation in an iteration of the dataflow graph to the end of the last operation in the same iteration.
- *Cycle time* – The delay between similar events in successive iterations of the dataflow graph.

Since the dataflow graph has multiple threads of computation which do not necessarily start or end simultaneously (see figure 9a), the loop delay is not the same as the cycle time. The inverse of the cycle time gives the rate at which successive values of intermediate variables in the computation are generated by the system. However, if the values of all variables at the end of $n$ iterations are needed, then the total time required is $((n - 1) \cdot \text{cycle time} + \text{loop delay})$.

To compute the loop delay, the timing constraint graph for a generic iteration (as opposed to a specific iteration, such as the first iteration) of the dataflow graph is constructed. Since there are multiple parallel threads of computation, this timing constraint graph has multiple source events. The starting time of an iteration is represented by taking the minimum of the times of occurrence of all source events. A min type event, StartOfIter, is therefore added to the timing constraint graph. Similarly, the end of computation in an iteration is represented by taking the maximum of ending times of all threads. In this case, a max type event, EndOfIter, is added. Bounds of the time separation between StartOfIter and EndOfIter give the best and worst-case loop delays of the dataflow graph. Note that both min and max type constraints are needed to model the loop delay.

To estimate the cycle-time, the time separation between the start of two consecutive iterations of the dataflow graph is computed. Two generic iterations of the dataflow graph are considered. The start of one iteration has already been modelled above. The start of the next iteration is modelled similarly by taking the minimum start times of the threads in the next iteration. Another min-type event, called StartOfNextIter, is therefore added to the timing constraint graph. Bounds of the time separation between StartofIter and StartOfNextIter give estimates of the best case and worst case cycle time of the system. Note that "cycle time" is sometimes used to denote the average time for $n$ iterations. If such a metric is desired, one must construct the timing constraint graph for $n$ iterations and determine the time separation between similar events spaced $n$ iterations apart. This separation divided by $n$ then gives the average cycle time.

6.4b *Timing constraints:* Timing constraints in the DiffEq datapath consist of setup, hold and minimum clock pulse-width constraints of registers, and setup constraints of domino circuits. In order to check register timing constraints, we examine every instance of a register loading data during a generic iteration of the dataflow graph. Setup-time violations are checked by computing the minimum time separation from the stabilization of the data inputs to the latching edge of the clock signal. If this separation exceeds the setup-time of the register, no setup-time violation can occur. Similarly, register hold-time constraints are checked by computing the minimum time separation from the latching edge of the clock to the subsequent change of the input data. Finally, the minimum clock pulse-width is obtained by determining the minimum time separation from the latching edge of the clock to the subsequent transition on the clock signal.

To verify the timing of a domino circuit, we consider every instance of the circuit performing some computation during a generic iteration of the dataflow graph. We determine the minimum time separation between the stabilization of all inputs and the start of the evaluation phase (InputsStable to A2Prech$-$ in figure 11b). A negative value of this separation indicates that the data inputs can potentially change after evaluation begins. This can lead to accidental discharge of the internal nodes, resulting in incorrect outputs. Note that if the input data lines are guaranteed to rise monotonically, the internal nodes cannot be accidentally discharged even if the inputs stabilize during the evaluation phase. However, the monotonic rising constraint is often too restrictive; hence ensuring that the data inputs stabilize before evaluation begins is a robust way of ensuring the correct operation of the domino circuit.

Finally, we consider verifying timing constraints in the inter-controller communication protocol. In DiffEq, these constraints arise from XBM requirements. For example, in the state transition diagram of MUL2Ctrl (figure 10a), rising transitions on both A2M and A1M enable the $S_0 \rightarrow S_1$ state transition and trigger a falling transition on M2prech. XBM semantics require that A2M subsequently remain high until the $S_1 \rightarrow S_2$ state transition has occurred and M2prech and M2A2 have risen. This gives rise to the constraint that the minimum time separation from each of M2A2$+$ and M2prech$+$ to A2M$-$ must be at least 0. Other timing constraints in the inter-controller communication protocol of DiffEq are of a similar nature, i.e., they are needed to ensure that an event always occurs some time after another event. Such constraints can be easily formulated in terms of time separations between events, as explained above. Note, however, that with non-XBM controllers, inter-controller protocol constraints may be complex, and it might not be possible to represent them simply as time separations between events.

## 6.5 *Experimental results*

We have manually constructed a cyclic timing constraint graph modelling the behaviour of DiffEq from a Register Transfer Level (RTL) description of the system. Since some time

**Table 4.** Timing verification results for typical-case SPICE delays with ±5% (top table) and ±10% (bottom table) variations.

All times are in ns. The entries marked with a ***** were obtained from timing verification experiments in which the delay of the buffer driving A1M was deliberately reduced

| | Event1 ($e_1$) | Event2 ($e_2$) | Required min. ($t_2 - t_1$) | Computed $[d, D]$ $d \leq (t_2 - t_1) \leq D$ | Timing violations? |
|---|---|---|---|---|---|
| Register | A1done+ | LA+ | Setup-time ($t_{su}$) | [1.1875, 1.3125] | None if $t_{su} < 1.1875$ |
| Timing | LA+ | A1done− | Hold-time ($t_h$) | [3.5135, 4.1465] | None if $t_h < 3.5135$ |
| Domino | DataStable_$M1$ | M1prech− | 0 | [0.7215, 2.2620] | None |
| Logic | DataStable_$A1$ | A1prech− | 0 | [0.9520, 1.4880] | None |
| Timing | DataStable_$A2$ | A2prech− | 0 | [2.4360, 3.5240] | None |
| | *****DataStable_$M14$ | M1prech− | 0 | [−0.2285, 2.2620] | **Potential violation** |
| Controller | M2prech+ | A2M− | 0 | [2.2465, 9.8465] | None |
| Protocol | M2A2+ | A2M− | 0 | [2.4985, 10.0175] | None |
| Timing | M2prech− | A1M− | 0 | [15.8785, 23.2325] | None |

| | Event1 ($e_1$) | Event2 ($e_2$) | Required min. ($t_2 - t_1$) | Computed $[d, D]$ $d \leq (t_2 - t_1) \leq D$ | Timing violations? |
|---|---|---|---|---|---|
| Register | A1done+ | LA+ | Setup-time ($t_{su}$) | [1.1250, 1.3750] | None if $t_{su} < 1.1250$ |
| Timing | LA+ | A1done− | Hold-time ($t_h$) | [3.1970, 4.4630] | None if $t_h < 3.1970$ |
| Domino | DataStable_$M1$ | M1prech− | 0 | [0.1930, 3.2330] | None |
| Logic | DataStable_$A1$ | A1prech− | 0 | [0.6750, 1.7560] | None |
| Timing | DataStable_$A2$ | A2prech− | 0 | [1.8920, 4.0680] | None |
| | *****DataStable_$M1$ | M1prech− | 0 | [−0.7070, 3.2330] | **Potential violation** |
| Controller | M2prech+ | A2M− | 0 | [2.0430, 11.5430] | None |
| Protocol | M2A2+ | A2M− | 0 | [2.3670, 11.7050] | None |
| Timing | M2prech− | A1M− | 0 | [14.8870, 25.2550] | None |

separations of interest involved events in two consecutive iterations of the dataflow graph, cutsets $C_1$ and $C_2$ were chosen such that: (a) subgraph $G_0$ models the behaviour of the system from Reset to the end of the second iteration, and (b) subgraph $G_1$ models two subsequent iterations of the dataflow graph.

Nominal delays of the circuit components were obtained from SPICE simulations under typical-case conditions (22°C, 3.3V and typical-case parasitics). A percentage variation, such as ±10%, was then introduced to obtain bounds of the delay intervals. For datapath components, delays for the best-case and worst-case data were also taken into account when determining the delay bounds.

Subgraph $G_1$ has 288 events. During phase I, bounds of the time separation of 129 pairs of events were monitored to check for timing constraint violations and to estimate bounds

**Table 5.** Performance analysis results. All times are in ns.

| | | |
|---|---|---|
| Loop delay | 0% variation | [37.4600, 51.5800] |
| | 5% variation | [35.5870, 54.8060] |
| | 10% variation | [33.7140, 59.3944] |
| Cycle time | 0% variation | [35.2200, 45.8400] |
| | 5% variation | [33.4590, 48.7790] |
| | 10% variation | [31.6980, 54.1580] |

of performance metrics. Phase I converged after 2 iterations and took 0.46s on a 150 MHz MIPS R5000 machine with 128 MB of memory [3]. In comparison, the symbolic performance analysis technique of Xie & Beerel (1997) took 16.5 minutes on a SPARC-20 with 64 MB of memory to analyse the performance of the same system. In phase II, the behaviour of DiffEq from Reset till the end of the fourth iteration of the dataflow graph was modelled and analysed. This graph contains 680 events, and bounds of the time separation of 441 pairs of events were monitored to check for timing constraint violations. This phase of analysis took 1.37s on the same computing platform as described above.

Since algorithm AcyclicApproxSep, used in both phases of the analysis, produces conservative results in the worst case, bounds computed in phases I and II could be inexact. For purposes of comparison, the analysis with $\pm 5\%$ delay variations was repeated with McMillan & Dill's (1992) branch-and-bound algorithm (an exact algorithm for acyclic timing constraint graphs) used in place of the AcyclicApproxSep algorithm. All the bounds were found to be identical to those obtained with AcyclicApproxSep. However, the complete analysis (phases I and II) using McMillan & Dill's (1992) algorithm took more than 11 hours on the same computing platform.

We present two subsets of our timing verification results, corresponding to $\pm 5\%$ and $\pm 10\%$ variations of delays, in table 4. Each entry lists a pair of events, the required minimum separation between these events, and the computed bounds of the separation during phases I and II (lower and upper bounds shown are the minimum of the lower bounds and the maximum of the upper bounds obtained in the two phases). Table 5 shows the loop delay and cycle time of DiffEq obtained with 0%, $\pm 5\%$ and $\pm 10\%$ variations in delays. The 0% entries indicate the loop delay and cycle time variations solely due to data-dependent delay variations.

Experiments have also been performed to determine how the correctness of the design depends on the delay of the A1M signal (an inter-controller synchronization signal) in figure 9b. This was motivated by feedback from the designers, who had observed from extensive SPICE simulations that in a preliminary version of the design, where the delay of the buffer driving A1M was small, the circuit could potentially malfunction. To investigate this, experiments were performed in which the delay of the buffer driving the A1M signal was deliberately reduced. The computed bounds immediately indicated potential setup-time violations of domino circuits in the system. The entries marked with a ***** in table 4 show some of these results. Note that the current analysis uncovered the potential errors within a few seconds, whereas it took several hours of SPICE simulations to detect the same problems. This demonstrates the practical utility of the approach.

[3]The time reported for "diffEq" in table 1 of § 4.3 is for one iteration of phase I

## 7. Drawbacks and limitations

Despite its advantages, there are drawbacks and limitations of the proposed method. These are summarized below.

- Currently, timing constraint graphs are manually constructed from descriptions (e.g. RTL description) of the system behaviour. For large and complex systems such as DiffEq, this is a tedious and error-prone task. Additional research on automating generation of timing constraint graphs from standard system descriptions needs to be pursued in order to facilitate widespread use of the proposed methodology. Note, however, that constructing a timing constraint graph from a standard system description is expected to be considerably easier, in general, than verifying complex timing constraints between events.
- Choice or conditional behaviour is an important feature of many practical asynchronous systems. Since we can neither model choice with timing constraint graphs, nor extend our algorithms in a straightforward way to deal with choice without considering all possibilities explicitly (exponential blowup in the worst-case), this is a severe limitation of the current work.
- Even for systems without choice, property **P4** in the definition of tightly-coupled systems (Definition 1) restricts the types of systems that can be analysed to those with few min and mostly max constraints. Yet another restriction is that the delays of all components, both in the system and its environment, must be finitely bounded. The second restriction makes it difficult to model interface circuits, where, for example, a system might interact with an environment with unknown delay characteristics. However, both property **P4** and finite edge delays are needed to ensure that finite bounds on the long-term time separation of events can be obtained in pseudo-polynomial time even without any knowledge of the initial system behaviour. This may be seen from the proofs of theorem 2 and corollary 1, which, in turn, are needed to prove lemma 2 and theorem 3. Systems satisfying properties **P1** through **P3**, but not **P4**, or systems with infinite edge delay bounds may still be analysed using the proposed algorithm. However, there is no guarantee that the computed bounds will be finite. Since infinite bounds are not very useful in general, the usefulness of analysing such systems with the proposed algorithm is unclear.
- Although the iterative refinement in phase I of algorithm CyclicApproxSep can be terminated after a pre-set number of iterations, it is usually desirable to let the analysis converge since that gives tighter bounds by the monotonic convergence property. A drawback of the current method is that for pathological cases, the number of iterations required to converge may depend on edge delay bounds, and hence, can be large.
- Each event in a timing constraint graph is associated with either a min or a max type constraint, but not both. However, in certain systems, the time of occurrence of an event depends on the min of the times of some of its predecessors and on the max of the times of other predecessors. In addition, linear constraints may exist between the time of occurrence of the current event and those of other events. Unfortunately, systems with multiple types of constraints associated with the same event cannot be modelled or analysed with the proposed technique.

## 8. Conclusion

In this paper, we presented two algorithms for timing analysis of asynchronous systems without choice or conditional behaviour.

(1) A polynomial-time algorithm for computing approximate bounds of time separation of events in acyclic timing constraint graphs, and
(2) a pseudo-polynomial time algorithm for computing the same bounds in a class of cyclic timing constraint graphs.

Both algorithms are capable of analysing systems with both min and max type constraints without incurring exponential complexity. This makes our algorithms more efficient and more general in their applicability compared to algorithms proposed earlier by Amon *et al* (1992), Hulgaard *et al* (1995) and Myers & Meng (1993). Although the results produced by our algorithms are conservative in the worst-case, experiments indicate that they are fairly accurate in practice. This is clearly borne out in the table of results for the acyclic algorithm. While we have described the results of applying the CyclicApproxSep algorithm to only DiffEq, application of the algorithm to a few other examples adapted from the literature (Amon 1993; Hulgaard 1995; Myers 1995) (which could not be described here due to lack of space) also yielded exact results. This suggests that carefully designed algorithms for efficient and approximate timing analysis of asynchronous systems is a promising alternative to more expensive exact techniques. Such techniques are also useful in design-verify-redesign environments where multiple passes of analysis may be required. Alternatively, efficient and approximate techniques can also be used as fast filters to narrow down the set of potential timing problems in large and complex designs. We believe that such techniques will play an important role in helping asynchronous circuits and systems gain wider acceptance.

While efficient analysis of asynchronous and concurrent systems was the primary motivation behind the work presented in this paper, one can also envisage applying the same techniques to synchronous systems in which different components interact with each other. The delays involved in such an application, e.g., the delay for a component to react in response to a message from another component, could be expressed in terms of the number of clock ticks involved. Hence, all delays would be integer valued. The analysis techniques presented above, however, continue to remain applicable for computing time separation between events in such communicating synchronous systems. Needless to say, the computed time separations would also be in terms of the number of clock ticks.

We thank Julio Arceo of Qualcomm Inc., San Diego, CA for helping us with SPICE simulations. We also thank IEEE for granting us permission to use material from our earlier publication (Chakrabarty *et al* 1999). We would like to thank the anonymous reviewers for their comments, which helped in improving the paper.

This work was supported by gifts from Sun Microsystems and Intel Corporation.

## References

Alur R, Henzinger T A 1992 Logics and models of real-time: A survey. In *Real time: Theory in practice. Lecture Notes in Computer Science #600* (eds) J W de Bakker, K Huizing, W P de Roever, G Rosenberg (Berlin: Springer-Verlag) pp. 74–106

Amon T 1993 *Specification, simulation and verification of timing behavior*. Ph D thesis, University of Washington, Seattle, WA

Amon T, Borriello G 1992 An approach to symbolic timing verification. In *Proceedings of the ACM/IEEE Design Automation Conference* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp. 410–413

Amon T, Hulgaard H 1999 Symbolic time separation of events. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 83–93

Amon T, Hulgaard H, Borriello G, Burns S 1992 Timing analysis of concurrent systems. Tech. Rep. UW-CS-TR-92-11-01, University of Washington, Seattle, WA

Amon T, Borriello G, Hu D, Liu J 1997 Symbolic timing verification of timing diagrams using Presburger formulas. In *Proceedings of the ACM/IEEE Design Automation Conference* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 226–231

Berks R, Ebergen J 1997 Response time properties of linear pipelines with varying cell delays. In *Proceedings of the ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 179–188

Borriello G 1998 *A new interface specification methodology and its application to transducer synthesis*. Ph.D thesis, University of California at Berkeley, CA

Brzozowski J A, Gahlinger T, Mavaddat F 1991 Consistency and satisfiability of waveform timing specifications. *Networks* 21: 91–107

Burks T M, Sakallah K A 1993 Min-max linear programming and the timing analysis of digital circuits. In *Proceedings of the International Conference on Computer-Aided Design* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 152–155

Burns S M 1991 *Performance analysis and optimization of asynchronous circuits*. Ph D thesis, California Institute of Technology, Pasadena, CA

Chakraborty S 1998 *Polynomial-time techniques for approximate timing analysis of asynchronous systems*, Ph D thesis, Stanford University, CA

Chakraborty S, Dill D L 1997 Approximate algorithms for time separations of events. In *Proceeding of the IEEE/ACM Internation Conference on Computer-Aided Design* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 190–194

Chakraborty S, Dill D L, Yun K Y, Chang K-Y 1997 Timing analysis for extended burst-mode circuits. In *Proceedings of the Third International Symposium on Advanced Research in Asynchronous Circuits and Systems* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 101-111

Chakraborty S, Yun K Y, Dill D L 1998 Practical timing analysis of asynchronous circuits using time separations of events. In *Proceedings of the IEEE Custom Integrated Circuits Conference* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 455-458

Chakraborty S, Yun K Y, Dill D L 1999 Timing analysis of asynchronous systems using time separation of events. *IEEE Trans. Comput. Aided Design* 18: 1061–1076

Chou P, Borriello G 1995 Interval scheduling: Fine-grained code scheduling for embedded systems. In *Proceeding of the ACM/IEEE Design Automation Conference* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp. 462–467

Ebergen J, Berks R 1997 Response time properties of some asynchronous circuits. In *Proceedings of the 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 76–86

Gahlinger T 1990 *Coherence and satisfiability of waveform timing specifications*. Ph D thesis, University of Waterloo, Waterloo, Ontario

Girodias P, Cerny E, Older W J 1997 Solving linear, min and max constraint systems using CLP based on relational arithmetic. *Theor. Comput. Sci.* 173: 253–281

Gunawardena J 1994 Timing analysis of digital circuits and the theory of min-max functions. Tech. Rep. HPL-94-39, Hewlett-Packard Laboratory, Palo Alto, CA

Hulgaard H 1995 *Timing analysis and verification of timed asynchronous circuits*. Ph D thesis, University of Washington, Seattle, WA

Hulgaard H, Burns S M 1994 Bounded delay timing analysis of a class of CSP programs with choice. In *Proceedings of the 1st International Symposium of Advanced Research in Asynchronous Circuits and Systems* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 2–11

Hulgaard H, Burns S M, Amon T, Borriello G 1994 An algorithm for exact bounds on time separation of events in concurrent systems. Tech. Report, No. UW-CSE-94-02-02, Dept. of Computer Sci. Eng., Univ. of Washington, Seattle, WA

Hulgaard H, Burns S M, Amon T, Borriello G 1995 An algorithm for exact bounds on time separation of events in concurrent systems. *IEEE Trans. Comput.* 44: 1306–1317

Kudva P, Gopalakrishnan G, Brunvand E 1994 Performance analysis and optimization for asynchronous circuits. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Oct. 1994, pp 221–225

Lavagno L, Sangiovanni-Vincentelli A L 1994 Linear programming for optimum hazard elimination in asynchronous circuits. *J. VLSI Signal Process.* 7: 137-60

Lee T K 1995 *A general approach to performance analysis and optimization of asynchronous circuits*. Ph D thesis, California Institute of Technology, Pasadena, CA

Mavaddat F, Gahlinger T 1998 On deducing timing constraints in the verification of interfaces. *Formal Methods Syst. Design* 12: 223–239

McMillan K L, Dill D L 1992 Algorithms for interface timing verification. In *Proceedings of the IEEE International Conference in Computer Design: VLSI in Computers and Processors* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp. 48–51

Mller J, Lichtenberg J, Andersen H R, Hulgaard H 1999 Difference decision diagrams. Tech. Rep. IT-TR-1999-023, Department of Information Technology, Technical University of Denmark, Lyngby

Myers C J 1995 *Computers-aided synthesis and verification of gate-level timed circuits*. Ph D thesis, Stanford University, Stanford, CA

Myers C J, Meng T H-Y 1993 Synthesis of timed asychronous circuits. *IEEE Trans. VLSI Syst.* 1: 106–119

Nielsen C D, Kishinevsky M 1994 Performace analysis based on timing simulation. In *Proceedings of the ACM/IEEE Design Automation Conference* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp. 70–76

Sakallah K A, Mudge T N, Olukotun O A 1990 CheckTc and minTc: Timing verification and optimal clocking of digital circuits. In *Proceedings of the 1990 IEEE/ACM International Conference on Computer Aided Design* (New York: ACM Press)

Tofts C 1995 A compositional analysis of the performance of asynchronous pipelines. In *Proceedings of the ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems* (Los Alamitos, CA: IEEE Comput. Soc. Press)

Vanbekbergen P, Goossens G, De Man H 1992 Specification and analysis of timing constraints in signal transition graphs. In *Proceedings of the European Design Automation Conference* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 302–306

Walkup E A 1995 *Optimization of linear Max-Plus systems with application to timing analysis*. Ph D thesis, University of Washington, Seattle, WA

Walkup E A, Borriello G 1994 Interface timing verification with application to synthesis. In *Proceedings of the ACM/IEEE Design Automation Conference* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 106–112

Williams T E 1994 Performance of iterative computation in self-timed rings. *J. VLSI Signal Process.* 7: 17–31

Xie A, Beerel P A 1997 Symbolic techniques for performance analysis of timed systems based on average time separation of events. In *Proceedings of the 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp 64–75

Yen T-Y, Wolf W 1995 Performance estimation of real-time distributed embedded systems. In *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors* (Los Alamitos, CA: IEEE Comput. Soc. Press) pp. 64–69

Yen T-Y, Ishii A, Casavant A, Wolf W 1998 Efficient algorithms for interface timing verification. *Formal Methods Syst. Design* 12: 241–265

Yun K Y 1994 *Synthesis of asynchronous controllers for heterogeneous systems*. Ph D thesis, Stanford University, Stanford, CA

Yun K Y, Beerel P A, Vakilotojar V, Dooply A E, Arceo J 1998 The design and verification of a high-performance low-control-overhead asynchronous differential equation solver. *IEEE Trans. VLSI Syst.* 6: 643–655