Timing Analysis of Asynchronous Systems Using Time Separation of Events

Supratik Chakraborty, *Member, IEEE,* Kenneth Y. Yun, *Member, IEEE,* David L. Dill, *Member, IEEE*

Abstract— **This paper describes a pseudo-polynomial time algorithm for timing analysis of a class of choice-free asynchronous systems, called tightly-coupled systems, with both min and max type timing constraints and bounded component delays. The algorithm consists of two phases: (i) long-term behavior analysis, that computes bounds on the time separation of events after the system has run for a sufficiently long period of time, and (ii) startup behavior analysis, that computes time separations between events during an initial startup period after the system is powered up. The results of the analysis are conservative in the worst-case; nevertheless, they are exact in our experiments. To demonstrate the practical utility of the approach, an asynchronous differential equation solver chip has been modeled and analyzed using the proposed algorithm. We report results of datapath timing verification, inter-controller protocol timing verification and performance analysis of the chip using the proposed technique.**

I. Introduction

There is mounting evidence that asynchronous circuits are finding a niche in high-performance applications, such as Intel's asynchronous instruction length decoder [1], [2] and the asynchronous differential equation solver benchmark circuit [3]. Common traits in these systems are a high degree of concurrency, distributed control and implicit timing assumptions to hide the control overhead. In order to guarantee correct operation of these systems as well as to estimate their performance, efficient timing analysis techniques are needed. The problem is compounded by the fact that statistical variations in manufacturing and operating conditions result in uncertainties in component delays in a chip. Consequently, timing analysis techniques for asynchronous systems with uncertain delays are needed.

A central problem in the analysis of asynchronous systems is computing bounds on the time separation between events [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. Existing techniques typically focus on controller behavior and inter-controller communication protocols, that involve synchronization or max type timing constraints in addition to linear constraints between event times. Systems with max-only or max-and-linear constraints have been well-studied and efficient algorithms exist for computing time separations in these systems [8], [9], [15], [16], [17], [11], [14]. However, modern high-performance asynchronous systems (e.g., the system in [3]) make implicit timing assumptions in both the datapath and control to facilitate design of high-speed circuits. Therefore, it is important to model and analyze both datapath and controller behaviors. Modeling datapath behavior, however, requires the use of min type timing constraints, especially when modeling components at a low level of abstraction. Similarly, performance metrics, such as the loop delay of an iterative process, are best modeled with both min and max type timing constraints. Therefore, there is a need for extending existing techniques for max-only systems to systems with min and max constraints, while remaining efficient.

Unfortunately, the problem of computing time separation bounds in systems with both min and max constraints is known to be computationally intractable [8]. Earlier researchers have proposed worst-case exponential-time techniques for analyzing such systems [18], [8], [13], [19]. However, exponential-time techniques are not always well-suited in practice, e.g., in a design-analyze-redesign environment where multiple passes of analysis are often needed. More efficient timing analysis techniques that are approximate in the worst-case, but compute reasonably accurate results in practice may be more useful in such situations. Such techniques can also serve as filters to quickly narrow down the set of potential timing problems in a large design. The reduced set of potential problems can then be analyzed by more expensive exact techniques, leading to a pragmatic approach to timing analysis of large systems.

The above considerations motivate the current work. We describe an efficient algorithm for computing approximate bounds on the time separation of events in a class of asynchronous systems called *tightly-coupled systems*. Events in a tightly-coupled system may repeat over time, component delays can assume arbitrary values between specified bounds, and both min and max type timing constraints may be present. However, for efficiency of analysis, the system is assumed to be *choice-free*. Although this restriction is significant, we believe that a large and important class of applications can be modeled and analyzed as tightly-coupled systems. The results of such an analysis must, however, be interpreted carefully — they apply only during choice-free operation of the system. To demonstrate the practical utility of the approach, an asynchronous differential equation solver has been modeled and analyzed using the proposed algorithm.

The main contributions of this paper are summarized below.

- Design of an efficient algorithm for computing approximate bounds on the time separation of events in a class of choice-free systems with *both* min and max type timing constraints. For systems with integer-bounded delays, an upper bound on the time complexity is derived.
- Application of the above algorithm to verify timing constraints and estimate performance bounds of an asynchronous differential equation solver chip.

The remainder of this paper is organized as follows. Section II surveys related work on timing analysis of asynchronous systems. Section III describes cyclic timing constraint graphs as

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

S. Chakraborty is with Fujitsu Laboratories of America, Inc., Sunnyvale, CA 94086, USA.

K. Y. Yun is with the Electrical and Computer Engineering Dept., University of California, San Diego, La Jolla, CA 92093-0407, USA.

D. L. Dill is with the Computer Science Dept., Stanford University, Stanford, CA 94305, USA.

a formalism for representing timing constraints in systems with repeated events and gives a characterization of tightly-coupled systems. Section IV describes an efficient algorithm for computing approximate bounds on the time separation of events in tightly-coupled systems. Section V describes application of the algorithm to verify timing constraints and estimate performance bounds of an asynchronous differential equation solver chip. Section VI describes the limitations of the proposed technique. Section VII concludes the paper.

II. Related Work

Burns [20] defined the cycle period of an asynchronous system as the asymptotic average time separation between consecutive occurrences of the same event, and proposed a polynomial-time algorithm for computing the period of max-only systems with fixed component delays. Subsequently, Lee [21] extended this work to systems with disjunctive constraints, of which min constraints are a special case. The worst-case complexity of Lee's technique is exponential in the size of the system. Gunawardena [22] gave a theoretical framework for analyzing systems of min and max constraints with fixed delays. He studied the periodic behavior of such systems and gave an analytic formula for the period. Several other researchers have also modeled asynchronous circuits as systems of repeated events with varying delay assumptions, and proposed different techniques for computing their performance metrics. A few representative works are those of Williams [23], Nielsen and Kishinevsky [24], Tofts [25], Kudva, Gopalakrishnan and Brunvand [26], Xie and Beerel [27] and Ebergen and Berks [28], [29].

McMillan and Dill [8] showed that finding exact bounds on the time separation of events is computationally intractable when both min and max timing constraints are present and component delays are uncertain but bounded. They proposed a polynomial-time algorithm for analyzing systems of non-repeated events with only max constraints, and gave a pseudo-polynomial time algorithm for systems with max and linear constraints. They also described a branch-and-bound algorithm, with worst-case exponential running time, for analyzing systems of min, max and linear constraints. Vanbekbergen et al. [9] proposed a cubic time algorithm for max-only systems with non-repeated events. Burks and Sakallah [19] described a branch-and-bound technique and a mixed integer linear program formulation for analyzing systems with min, max and linear timing constraints; however, both their techniques have worst-case exponential running time. In her dissertation [11], Walkup described an algorithm for analyzing systems of linear and max constraints, with applications to interface timing verification and interface logic synthesis. Yen et al. [14] have described another algorithm for interface timing verification in systems with max and linear constraints. The algorithms of both Walkup and Yen apply to systems with non-repeated events, and are *conjectured* to run in polynomial-time. The exact complexity of the max and linear problem is, unfortunately, not yet known.

Myers and Meng described a polynomial-time algorithm for approximate timing analysis of max-only systems with repeated events and bounded delays [15]. In their method, a cyclic graph representation, called an *event-rule* system, is used to represent temporal dependencies between events. Their algorithm effectively unfolds this graph into an infinite acyclic graph and analyzes a finite subgraph to compute approximate bounds on the time separation between events. Amon, Hulgaard, Burns and Borriello [16], [17] considered systems of repeated events with only max constraints, and proposed an algebraic technique for computing exact bounds on the time separation of events. Their technique implicitly unfolds a cyclic graph into an infinite acyclic graph and then uses algebraic techniques to determine the maximum time separation between a pair of events, maximized over all unfoldings. The worst-case complexity of their technique is exponential in the size of the system description. Hulgaard and Burns[30], [13] also extended this work to systems with certain types of choices. Among other related works, Amon and Borriello [18] have described a symbolic timing verification technique based on constraint logic programming (CLP). Girodias, Cerny and Older [31] have described another timing verification technique based on CLP with relational interval arithmetic. Symbolic timing verification using Presburger formulas has been studied recently by Amon et al. [32]. These techniques, however, have worst-case complexity at least exponential in the size of the system description. Chakraborty et al. [33] have described a polynomial-time technique for obtaining conservative estimates of timing constraints required for correct operation of a class of asynchronous circuits. In [3], a technique for uncovering timing bugs in an asynchronous differential equation solver chip has been reported.

The work that comes closest to the present work is that of Myers and Meng [15]. However, there are important extensions to and differences from their work, as summarized below.

- The proposed algorithm can be used to analyze a class of systems with both min and max timing constraints. In contrast, Myers and Meng's algorithm is applicable to max-only systems.
- For systems with finite component delays, it is shown that after the system has run for a sufficiently long period of time, finite bounds on the time separation of events can be obtained even without any knowledge of the the behavior of the system immediately after power-up. Although this applies to Myers and Meng's algorithm as well, they do not prove that the bounds are finite.
- The proposed algorithm consists of an iterative refinement phase, in which the computed time separation bounds monotonically approach their asymptotic values with every iteration. This is called *monotonic convergence* of the bounds. Myers and Meng do not study similar properties of the computed bounds.
- A notion of convergence of analysis is defined, and an analytic upper bound on the number of iterations until converegence is derived for systems with integer-bounded delays. The work of Myers and Meng does not give such an analytic bound.

III. Problem Representation and Formalization

This section describes cyclic timing constraint graphs as a formalism for representing temporal dependencies in choice-free systems with repeated events. Tightly-coupled systems are characterized, and the time separations problem is then formalized.

A. Cyclic timing constraint graphs

A cyclic timing constraint graph is a directed, labeled graph $G = (V, E)$. In general, the graph has two components – an

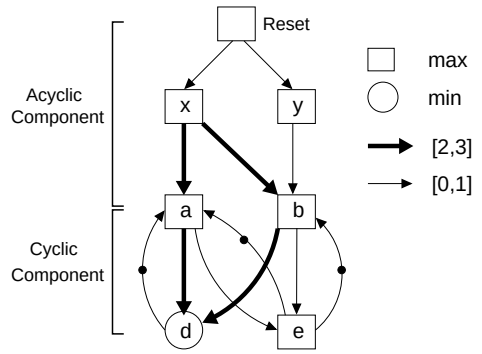


Fig. 1. A cyclic timing constraint graph. Square vertices represent max events, circles represent min events. Edge delays are shown in the legend.

acyclic component modeling the behavior of the system immediately after it is powered up or reset, and a cyclic component modeling the subsequent repeated behavior (see Fig. 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 et al [16], [17] 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. It is assumed that there exists a unique Reset event, represented by a vertex with no incoming edges (source vertex), that represents the start of operation of the system. Every vertex is associated with a min or max operator, but not both, that determines how the time of occurrence of the corresponding event(s) depends on those of other events. In this paper, a vertex with a min operator is represented by a circle, and one with a max operator is represented by a square (see Fig. 1). For consistency of notation, a vertex is referred to as an event in the remainder of this paper.

Every edge, $\langle u, v\rangle$, from u to v represents a timing constraint between events u and v, and is labeled with a delay variable, $\delta_{u,v}$. This represents the delay in the propagation of the effect of event u to the component that generates event v. The delay $\delta_{u,v}$ can assume any value within fixed lower and upper bounds, $d_{u,v}$ and $D_{u,v}$. Every edge is also 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. In contrast, 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 Fig. 1. For clarity, edges with different delay intervals are represented by different types of arrows in this paper (see, for example, the legend in Fig. 1).

Let τ_{v_k} denote the time of occurrence of event v_k. If $preds(v_k)$ denotes the set of events with an edge to v_k, and if v_k is a max-type event, then

$$\tau_{v_k} = \max_{u \in preds(v_k)} (\tau_u + \delta_{u,v_k}),\ d_{u,v_k} \leq \delta_{u,v_k} \leq D_{u,v_k}.$$

For brevity, this will henceforth be written as: $\tau_{v_k} = \max_{u \in preds(v_k)} (\tau_u + [d_{u,v_k}, D_{u,v_k}])$. Similarly, for a min-type event, the expression is: $\tau_{v_k} = \min_{u \in preds(v_k)} (\tau_u + [d_{u,v_k}, D_{u,v_k}])$. Thus, if every delay variable is assigned a value within the corresponding bounds, the time of occurrence of every event is well-defined, and the occurrence of one event cannot disable the occurrence of any other event. Hence, systems with conflict or choice cannot be modeled by cyclic timing constraint graphs.

Following the terminology of Amon et al. [16], [17] and Myers et al. [15], a timing constraint graph is said to be *well-formed* if the following conditions are satisfied: (a) every cycle has at least one marked edge, and (b) for every event v in the cyclic component, there exists at least one cycle with exactly one marked edge that contains v. Condition (a) ensures that no event is deadlocked, waiting for itself to occur. Condition (b) ensures that the time of occurrence of v_k is well-defined for every occurrence index k. All cyclic graphs considered in this paper are assumed to be finite and well-formed.

Let $G = (V, E)$ be a finite, well-formed cyclic timing constraint graph. The timing constraints represented by G can be equivalently represented by an 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. Borrowing the terminology used by previous researchers [16], [15], [17], G^* is called the *unfolded graph* of G.

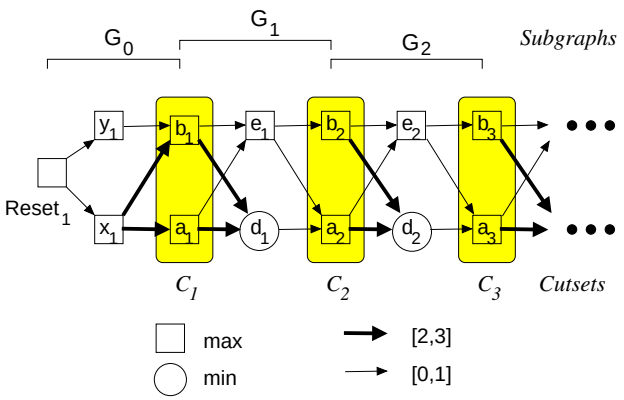


Fig. 2. Unfolded graph of cyclic timing constraint graph in Fig. 1.

Example 1 Consider the infinite acyclic graph in Fig. 2 and let us ignore the shaded boxes around vertices a_i, b_i and the annotations regarding subgraphs and cutsets for now. The infinite acyclic graph is the unfolded version of the cyclic timing constraint graph in Fig. 1. The timing constraints represented by the

infinite graph are summarized below.

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

B. Tightly-Coupled Systems

Given an unfolded graph, $G^* = (V^*, E^*)$, of a well-formed cyclic timing constraint graph, a path from u to v is defined as a sequence of one or more events $(u, \ldots x, y, \ldots v)$ such that $\langle x, y \rangle$ is an edge in E^* for every pair of consecutive events x and y in the sequence. By definition, (u) is a path from event u to itself. Event v is said to be *reachable* from u if there exists a path from u to v; otherwise, v is *unreachable* from u. By definition, every event is reachable from itself.

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 Fig. 2 is a cutset of the infinite acyclic graph.

Definition 1 *A cyclic timing constraint graph is called* ***tightly-coupled*** *if the unfolded graph has a sequence of cutsets, C_i for all $i \geq 1$, that satisfy the following properties:*
P1: *There are finitely many events that are* not *reachable from C_1.*
P2: *There exists a finite, positive integer α 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 associated with max constraints.*

As an example, consider the unfolded graph in Fig. 2. The shaded sets of events, C_1, C_2 and C_3, represent the first three cutsets in an infinite sequence satisfying properties **P1** through **P4**. Therefore, the cyclic timing constraint graph in Fig. 1 is tightly-coupled.

The reader may consider property **P4** to be overly restrictive and wonder if removing min-type events, such as d_i in Fig. 2, keeps the temporal behavior of the rest of the system unchanged. However, it can be shown that removing the d_i's in Fig. 2 causes the maximum time separation from b_i to a_i to change, for all i greater than or equal to 2. Property **P4** is, in fact, equivalent to the requirement of strongly connected graphs in the max-only systems considered by Amon et al. [16], Hulgaard et al. [17] and Myers and Meng [15]. Indeed, the systems considered in these earlier works are tightly-coupled systems with only max type events.

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 completeness, subgraph $G_0 = (V_0, E_0)$ is defined as the component modeling the behavior 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 behavior modeled by G_0 is referred to as the *initial behavior* of the system, and the behavior modeled by G_i ($i \geq 1$) is referred to as *iteration* i of the system.

If α in property **P2** is chosen to be greater than 1 (see, for example, Fig. 7), subgraph G_i may include multiple occurrences of the same event. This can lead to confusion about the labeling of events. To keep the notation unambiguous, the events in the unfolded graph are relabeled 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 α in property **P2** reduces to 1.

C. The Problem

The time separation of events problem for tightly-coupled systems can now be formalized as follows:
Given subgraphs G_i ($i \geq 0$) of the unfolded graph,
1. *Determine upper bounds on the time separations of all ordered pairs of events in G_0.*
2. *Determine upper bounds on the time separations of all ordered pairs of events in G_i, maximized over all $i \geq 1$.*

IV. Analysis of Tightly-Coupled Systems

This section describes a pseudo-polynomial time algorithm for computing the time separation bounds described above. Since G_i is isomorphic to G_1 for all i greater than or equal to 1, it suffices to analyze subgraphs G_0 and G_1. The algorithm consists of two phases, as summarized below.

- The first phase uses a successive refinement approach to compute bounds on 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 detail in Section IV-A below. Since the bounds computed in this phase apply only after the system has run for some time, they are called *long-term time separation bounds* and this phase of analysis is called *long-term behavior 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 modeling the system behavior from Reset to the events in cutset C_K.

Bounds on the time separation of events in G_0 are obtained directly from the second phase. To obtain an upper bound on the time separation of a pair of events in G_i, maximized 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.

A. Phase I of Analysis

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

Let n denote the number of events in G_1. Bounds on the time separation of events in G_1 are represented using an $n \times n$ matrix, Δ. The entry in row u and column v, denoted $\Delta(u, v)$, represents an upper bound on the time separation from event u to event v. Since subgraph G_1 is acyclic, an algorithm for analyzing acyclic timing constraint graphs suffices to compute the entries of the Δ matrix. For the current work, the AcyclicApproxSep algorithm of Chakraborty and Dill [34], outlined in Fig. 3, is used for this purpose.[1] 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, in this work, we choose to decouple the analysis of the initial behavior from the long-term behavior analysis and assume that the initial behavior 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 invoked. As will be seen later, this ensures that the long-term behavior analysis terminates within a pseudo-polynomial number of steps, while producing finite, conservative bounds on the long-term time separation of events. Note that since no assumptions are made about the initial behavior of the system, the time separation bounds computed in this phase apply regardless of the actual initial behavior. Therefore, an upper bound computed in this phase may be viewed as maximized over all possible initial behaviors.

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 to be $+\infty$ as well. However, this is not the case in tightly-coupled systems, as shown below.

[1] Details of the algorithm may be found in [34].

```
/* G is an acyclic timing constraint graph; n = total no. of      */
/* events; m = no. of source events (no incoming edges).          */
/* 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.              */
/* Δ: an n × n matrix; Δ(i, j) = an upper bound of τ_j − τ_i.     */
/* Bounds on the time separation 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, Δ : matrix)
InitializeDelta(Δ);
for each i in m to (n − 1)
  1. ComputeInitialEstimates(i, Δ);
  2. for each j in 0 to i − 1
       if (event j is max type)
         (2a) Δ(i, j) =
                min(Δ(i, j), max_{k ∈ preds(j)}(Δ(i, k) + D_{k,j}));
         (2b) Δ(j, i) =
                min(Δ(j, i), min_{k ∈ preds(j)}(Δ(k, i) − d_{k,j}));
       else if (event j is min type)
         (2c) Δ(i, j) =
                min(Δ(i, j), min_{k ∈ preds(j)}(Δ(i, k) + D_{k,j}));
         (2d) Δ(j, i) =
                min(Δ(j, i), max_{k ∈ preds(j)}(Δ(k, i) − d_{k,j}));

InitializeDelta (Δ : matrix)
 for each i in 0 to (n − 1)
   for each j in 0 to (n − 1)
     if (i == j)    Δ(i, i) = 0;
     else if (i < m AND j < m)     /* Source events */
       Δ(i, j) = given upper bound on (τ_j − τ_i);
     else    Δ(i, j) = +∞;

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

Fig. 3. Algorithm for computing time separation of events in acyclic graphs, adapted from [34].

Theorem 1 *For every pair of events u and v in C_2, the value of $\Delta(u, v)$ computed by algorithm* AcyclicApproxSep *is bounded above by $L(v) - l(u)$.*

Proof: See Appendix. ■

Corollary 1 *For every pair of events u and v in C_2, the value of $\Delta(u, v)$ computed by algorithm* AcyclicApproxSep *is finite.*

Proof: Follows from Theorem 1 and the observation that $\Delta(i, j)$ cannot be $-\infty$, since the Δ matrix is initialized with $+\infty$ and 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 on the time separation of events in subgraphs G_i for $i \geq 1$. Although

the algorithm actually analyzes 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, Δ_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 on 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 on 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 the 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 on 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 analytic upper bound on the number of iterations required to converge can be derived. (see Lemma 2). 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 be in the range 10 to 20, the analysis will 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 Fig. 4 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 [34], is $O(n^2 \cdot p)$. The complexity of Phase I is therefore $O(K_{max} \cdot n^2 \cdot p)$.

In order to characterize the bounds computed in Phase I, the $\leq_{\mathrm{M}}$ relation for Δ matrices is defined as follows. Let Δ and Δ' be two $n \times n$ matrices. Δ is said to be bounded above by Δ', denoted $\Delta \leq_{\mathrm{M}} \Delta'$, 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, $\Delta_{i+1} \leq_{\mathrm{M}} \Delta_i$. This property is called **monotonic convergence** of the time separation intervals.*

Proof: See Appendix. ■

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: See Appendix. ■

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, Δ_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. AyclicApproxSep($G_{startup}, \Delta$);

Fig. 4. Timing analysis algorithm for tightly-coupled systems.

Theorem 2 *Suppose Phase I terminates after K iterations and K is 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 behavior of the system.*

Proof: See Appendix. ■

B. Phase II of Analysis

Suppose Phase I terminates after K iterations. By Theorem 2, the bounds in Δ_K apply to events in every subgraph G_i for $i \geq K$. To compute bounds on 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 analyzed using the AcyclicApproxSep algorithm. This is shown in Phase II of the pseudo-code in Fig 4.

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 $O(m^2 + K_{max} \cdot n^2)$ time. The resulting graph has at most $m + K_{max} \cdot n$ events. If p denotes the maximum number of predecessors of an event, algorithm AcyclicApproxSep takes $O((m + K_{max} \cdot n)^2 \cdot p)$ time to analyze this graph. Therefore, the complexity of Phase II is $O((m + K_{max} \cdot n)^2 \cdot p)$. To compute bounds on the time separation of a pair of events in G_i, maximized 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 $O(K_{max} \cdot n^2)$ comparisons. Therefore, the complexity of the entire procedure is $O((m + K_{max} \cdot n)^2 \cdot p)$.

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 behavior of the system. The reader may wonder why we need to start the analysis assuming no knowledge of the initial behavior. However, as may be seen from the basis case of the proof of Lemma 1 (see Appendix), 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 are found to be exact.

C. Examples

Let us now apply the above algorithm to some examples. This illustrates some of the properties described above.

Example 2 Consider a two-processor system processing tasks from a task queue. Each task is divided into two subtasks; thus, there can be up to two subtasks at the head of the queue at any time. All subtasks at the head must belong to the same task. Thus, if one subtask of task T_i is scheduled on a processor, but the other is not, there will be only one subtask at the head of the queue.

Subtasks are assigned to the processors by a task manager as follows. Initially, each subtask of the first task is assigned to a different processor. Processing a subtask produces an intermediate result; intermediate results from the subtasks are then combined to generate the final result, which consists of two components. Each component of the final result is computed by a different processor. As soon as one component is computed, it is sent to the task manager. The task manager then schedules the next subtask from the head of the queue (if there are two subtasks at the head, one is arbitrarily chosen) on the idle processor, without waiting for the other component of the current result. Thus, subtasks of multiple tasks may be simultaneously active in the two-processor system.

To prevent subtasks of different tasks from interfering with each other, some bookkeeping is done for each task. Assume that for each task, this is done once immediately after the first subtask enters the system, and then again after the first intermediate result is produced. Assuming infinite tasks in the task queue, we wish to model the behavior of this system and determine bounds on the *latency* of a task, defined as the time from the first subtask entering the system to the final component of the result being generated.

Suppose the processing time of each subtask varies between 4 and 5 time units, each bookkeeping operation takes between 1 and 2 time units, and the time to combine intermediate results to generate components of the final result takes 2 to 3 time units. Furthermore, assume that the first subtask of the first task is scheduled immediately after Reset, but the second subtask is scheduled 20 time units after Reset. Fig. 5 shows the cyclic timing constraint graph modeling the system behavior. Note that pseudo-events EntryOfEarly(Late)ArrivingSubtask and CompletionOfEarly(Late)EndingSubtask have been introduced to distinguish between early and late events.

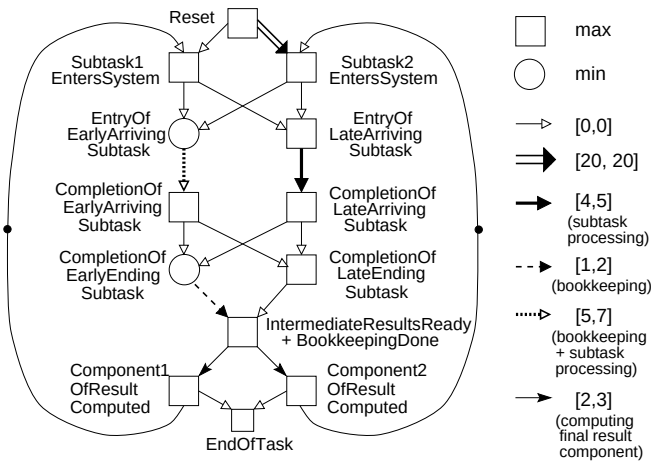


Fig. 5. Processing tasks with 2 processors. Edge delays are shown in the legend.

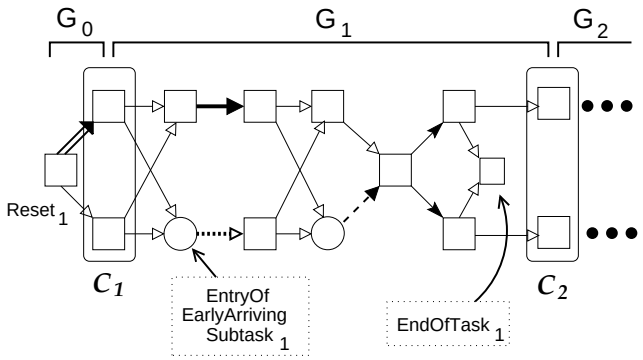


Fig. 6. Unfolded graph of Fig 5. For clarity, the labels of some events are omitted. Edge delays are shown in the legend of Fig. 5.

Fig. 6 shows the unfolded graph of Fig. 5 with subgraphs G_0 and G_1. The separation from EntryOfEarlyArrivingSubtask$_i$ to EndOfTask$_i$ in G_i gives the latency of task i, denoted T_i. On applying algorithm CyclicApproxSep to this graph, Phase I converges after 2 iterations and gives 7 and 11 as bounds on the latency of a task. By Theorem 2, these bounds apply to all subgraphs G_i, and hence to all tasks T_i, with $i \geq 2$. Phase II (analysis of G_0 and G_1) indicates that the latency of the first task, T_1, lies between 26 and 28 time units. This large latency is due to the skew in the starting times of the subtasks of T_1.

Note that the bounds computed in Phase I ($[7, 11]$) do not apply to subgraph G_1, but to all subgraphs G_i with $i \geq 2$. Note also that since no knowledge of the initial behavior of the system is assumed during Phase I, the latency of tasks T_i for all $i \geq 2$ lie between 7 and 10*, regardless of the initial skew in the starting times of subtasks of* T_1.

Example 3 The next example, shown in Fig. 7a, is adapted from Amon et al's work [12]. This example demonstrates that

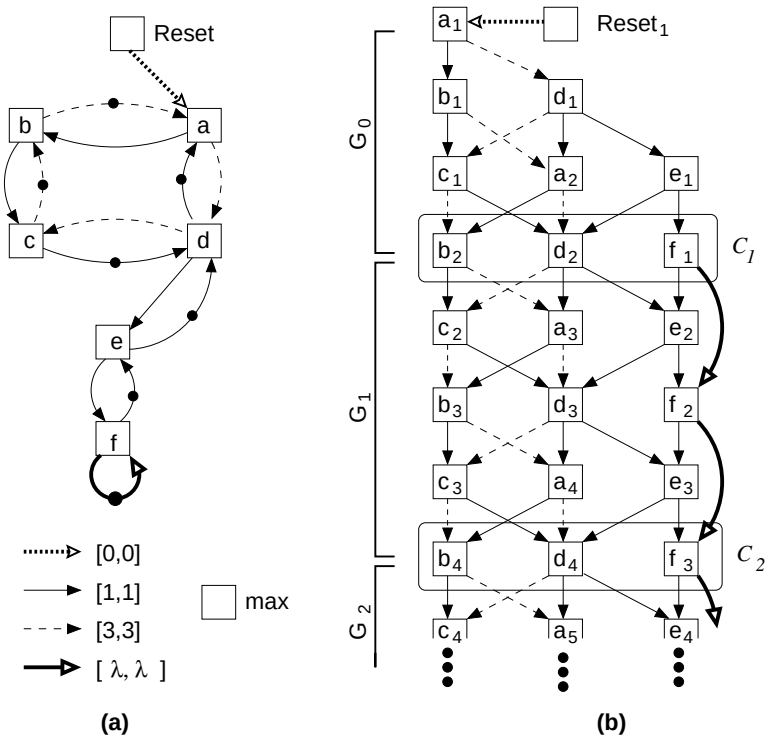


Fig. 7. Analyzing Amon et al.'s example [12]. Edge delays are shown in the legend.

If $\lambda = 6$:		If $\lambda = 9$:								
M^{even}	M^{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

TABLE I

TIME SEPARATION BOUNDS FOR SYSTEM OF FIG. 7. M^i DENOTES THE EXACT UPPER BOUND OF $\tau_{a_i} - \tau_{a_{i-1}}$ FOR $i \geq 2$.

the number of iterations required to converge depends on the edge delay bounds in general.

Let M^i denote the exact upper bound of $\tau_{a_i} - \tau_{a_{i-1}}$, for all i greater than or equal to two.[2] It has been shown by Amon et al. [12] that M^i depends on both i and λ, as indicated in Table I.

Fig. 7b shows subgraph G_1 of the unfolded graph. For clarity, the events in G_1 have not been relabeled to have occurrence index 1 — the relabeling 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 λ is 6, and $\Delta_2(a_3, a_4)$ 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 depend on the edge delay bounds in general.

From the above results and Theorem 2, we conclude that if λ equals 6, M^i is bounded above by 8 for all even i greater than or equal to 4. Similarly, if λ 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 λ is 6, and $\Delta_5(a_4, a_5)$ equals 9 when λ is 9. Therefore, M^i is bounded above by the same numbers for all odd i greater than or equal to 5. This matches the exact bounds computed by Amon et al.

[2] M^i in this notation equals Δ^{i-1} in the notation of Amon et al. [12].

V. APPLICATION: ANALYZING AN ASYNCHRONOUS CHIP

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

A. Chip Overview

Fig. 8, adapted from Yun et al. [3], 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, however, we are interested only in the system behavior as it iterates through the dataflow graph, so the termination condition is ignored. This, of course, implies that the computed re-

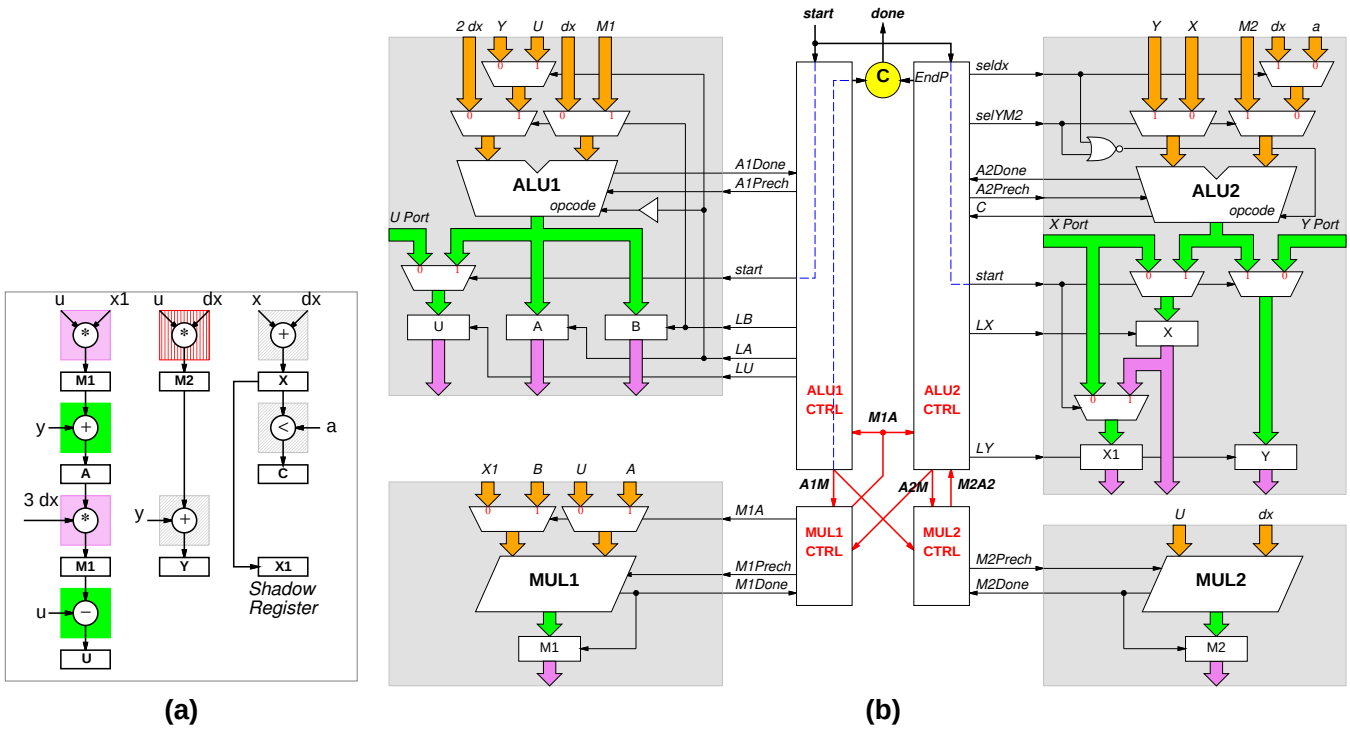


Fig. 8. (a) Dataflow graph; (b) Architecture of differential equation solver.

sults do not apply to the final iteration when the system chooses to terminate the computation. The behavior in the final iteration must be analyzed separately, and is not addressed in this paper. Details of the DiffEq design are described in Yun et al.'s paper [3].

The control logic of DiffEq is implemented using four *distributed* extended burst-mode (XBM) controllers [35]. These are labeled ALU1 CTRL, ALU2 CTRL, MUL1 CTRL and MUL2 CTRL in Fig. 8b. If we ignore the checks for the termination condition in their state transition diagrams, each controller exhibits choice-free behavior [3]. The controllers communicate with each other by means of a handshaking protocol that, unlike speed-independent or delay-insensitive designs, assumes safe timing bounds in signaling. The datapath is structurally identical to those found in synchronous designs, except for the completion-reporting logic associated with the ALU1, ALU2, MUL1 and MUL2 units. As in synchronous systems, the control is responsible for generating ALU opcodes, MUX selects, register load enables, and precharge/evaluate signals. Unlike conventional asynchronous circuits, there are implicit assumptions on the timing of datapath signals, which must be satisfied for the system to function correctly. For example, data inputs to domino circuits must stabilize before evaluation begins.

B. Modeling 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 can be obtained by the technique described by Chakraborty et al. [33] 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. The XBM semantics require that all *terminating* transitions [35] 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 behavior, 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, Fig. 9a shows the XBM state transition diagram of the MUL2 controller 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 thesis [35]. Fig. 9b shows how each state transition is modeled using timing constraint graphs. Note that A1M is a directed don't care during the state transition $S_1 \rightarrow S_2$, so by XBM semantics, the controller need not wait for a transition on A1M to change state from S_1 to S_2. This is reflected in Fig. 9b 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

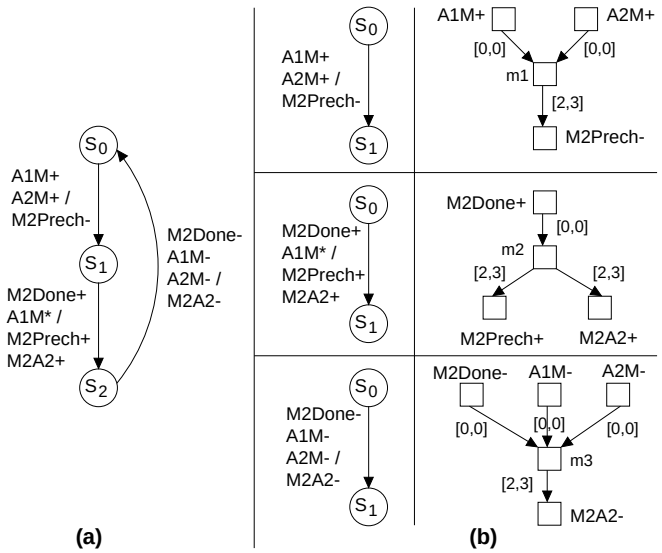


Fig. 9. Modeling temporal behavior of MUL2 CTRL. (a) MUL2 controller; (b) Timing constraint graph fragments (controller delay = $[2, 3]$).

for A1M to fall before changing states from S_2 to S_0. This is represented by drawing an edge from A1M$-$ to $m3$ in Fig. 9b.

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: MUL2 CTRL 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 MUL2 CTRL to MUL2 and back lies between 4 and 5 time units, an edge is drawn from M2prech+ to M2done$-$ and labeled with $[4, 5]$. Similarly, consider input event A2M$-$ in the graph fragment modeling $S_2 \rightarrow S_0$. This is an inter-controller synchronization event, generated by ALU2 CTRL and communicated to MUL2 CTRL. To model this communication, the corresponding events in the timing constraint graphs for ALU2 CTRL and MUL2 CTRL are connected by an edge labeled with the signal propagation delay from ALU2 CTRL to MUL2 CTRL. Continuing this process, the timing constraint graph fragments of the different controllers can be stitched together to obtain one larger graph modeling the behavior of the four interacting controllers during one iteration of DiffEq.

The above discussion concentrated on modeling 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 [20], signal graphs [24], 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 [21]), a min type constraint is used instead.

C. *Modeling datapath timing*

This section describes modeling techniques for two important datapath components. The behavior of other datapath components can be modeled similarly.

C.1 Arithmetic circuits designed in domino logic

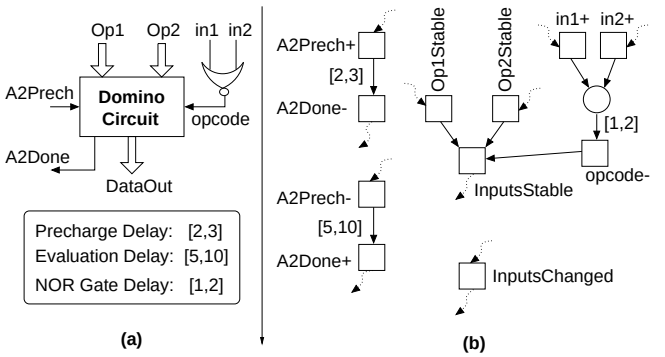


Fig. 10. Modeling a domino circuit. (a) Circuit; (b) Timing constraint graph: dotted edges represent connections to/from other parts of timing constraint graph. Unlabeled edges imply $[0, 0]$. Squares represent max-type events, circles represent min-type events.

Fig. 10a shows the ALU2 unit of DiffEq implemented in domino logic. The A2Done signal is used to notify other units that all internal nodes have been precharged (A2Done$-$) or that evaluation has completed (A2Done+). The signal value of opcode determines the type of operation (addition or subtraction) 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 behavior is modeled by the directed edge from A2Prech+ to A2Done$-$ in Fig. 10b.

The signal opcode at the output of the NOR gate in Fig. 10a falls as soon as one of its inputs rises. Therefore, the falling of opcode is modeled by a min type event (circle in Fig. 10b). This demonstrates the need for min type constraints when modeling 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 modeled as shown in Fig. 10b. The event InputsChanged in Fig. 10b represents a change in the value of one of the inputs after evaluation.

C.2 Edge-triggered registers

Fig. 11a shows register A, a positive edge-triggered register, clocked by the signal LA, in DiffEq (see Fig. 8b). For simplicity, let us assume that there are no asynchronous set/reset inputs. If the data inputs remain stable within the setup and hold-time window around the rising edge of LA (this is checked during timing verification), data is loaded when LA rises, and appears at the

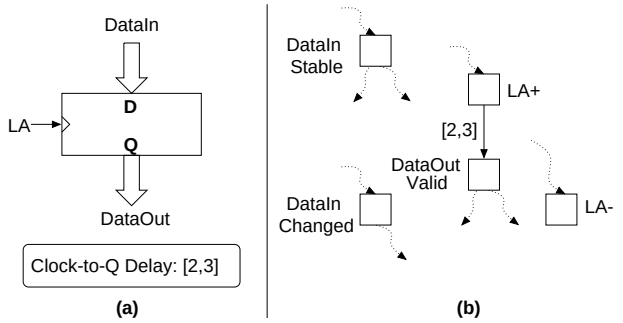


Fig. 11. Modeling an edge-triggered register. (a) Register; (b) Timing constraint graph.

output port after a "clock-to-Q" delay. This is modeled as shown in Fig. 11b. The event DataOutValid represents the appearance of latched data at the output port of the register. DataInStable and DataInChanged represent the stabilization of the data inputs prior to latching, and their subsequent change. The event LA− represents the falling of LA after the latching LA+ transition.

If a level-sensitive latch with the high clock phase as the active period was used in place of the edge-triggered register, DataOutValid would be a max type event with two incoming edges – one from DataInStable labeled with the "data-to-Q" delay, and the other from $LA+$ labeled with the "clock-to-Q" delay. This, of course, assumes that the setup and hold times are satisfied, which can be checked by computing the time separations from DataInStable and DataInChanged to the latching edge $LA-$.

D. 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. Note that since the CyclicApproxSep algorithm is conservative in the worst case, false timing violations may be reported and the estimated performance bounds may be pessimistic in the worst case.

D.1 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 Fig. 8a), 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 the ending times of all threads. In this case, a max type event, EndOfIter, is added. Bounds on 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 modeled above. The start of the next iteration is modeled 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 on the time separation between StartofIter and StartOfNextIter then 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.

The reader may argue that bounds on the loop delay or cycle time are not representative performance metrics; instead, average loop delay and cycle-time are. However, we feel that finding bounds on these metrics is useful because it allows one to investigate how variations in component delays affect the best case and worst case system performance. This is important if one is interested in performance guarantees, such as an upper bound on the delay for n iterations or the minimum rate at which results are computed.

D.2 Timing constraints

Timing constraints in the DiffEq datapath consist of setup, hold and minimum clock pulse-width constraints of registers, and setup-time 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 (DataInStable to LA+ in Fig. 11b). 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 (LA+ to DataInChanged in Fig. 11b). 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 same signal (LA+ to LA− in Fig. 11b).

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 mini-

	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 Timing	A1done+	LA+	Setup-time (t_{su})	[1.1875, 1.3125]	None if $t_{su} < 1.1875$
	LA+	A1done−	Hold-time (t_h)	[3.5135, 4.1465]	None if $t_h < 3.5135$
Domino Logic Timing	DataStable_M1	M1prech−	0	[0.7215, 2.2620]	None
	DataStable_A1	A1prech−	0	[0.9520, 1.4880]	None
	DataStable_A2	A2prech−	0	[2.4360, 3.5240]	None
	*DataStable_M1	M1prech−	0	[−0.2285, 2.2620]	**Potential violation**
Controller Protocol Timing	M2prech+	A2M−	0	[2.2465, 9.8465]	None
	M2A2+	A2M−	0	[2.4985, 10.0175]	None
	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 Timing	A1done+	LA+	Setup-time (t_{su})	[1.1250, 1.3750]	None if $t_{su} < 1.1250$
	LA+	A1done−	Hold-time (t_h)	[3.1970, 4.4630]	None if $t_h < 3.1970$
Domino Logic Timing	DataStable_M1	M1prech−	0	[0.1930, 3.2330]	None
	DataStable_A1	A1prech−	0	[0.6750, 1.7560]	None
	DataStable_A2	A2prech−	0	[1.8920, 4.0680]	None
	*DataStable_M1	M1prech−	0	[−0.7070, 3.2330]	**Potential violation**
Controller Protocol Timing	M2prech+	A2M−	0	[2.0430, 11.5430]	None
	M2A2+	A2M−	0	[2.3670, 11.7050]	None
	M2prech−	A1M−	0	[14.8870, 25.2550]	None

TABLE II

TIMING VERIFICATION RESULTS FOR TYPICAL-CASE SPICE DELAYS WITH $\pm 5\%$ (TOP TABLE) AND $\pm 10\%$ (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.

mum time separation between the stabilization of all inputs and the start of the evaluation phase (InputsStable to A2Prech− in Fig. 10b). 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 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 MUL2 CTRL (Fig. 9a), rising transitions on both A2M and A1M enable the $S_0 \rightarrow S_1$ state transition and trigger a falling transition on M2prech. The 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 constraints 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 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.

E. Experimental Results

We have manually constructed a cyclic timing constraint graph modeling the behavior of DiffEq from an RTL description of the system. Since some time separations of interest involve events in two consecutive iterations of the dataflow graph, cut-sets C_1 and C_2 are chosen such that: (a) subgraph G_0 models the behavior 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 are obtained from SPICE simulations under typical-case conditions (22°C, 3.3V and typical-case parasitics). A percentage variation, such as $\pm 10\%$, is then introduced to obtain bounds of the delay intervals. For datapath components, delays for the best-case and worst-case data are also taken into account when determining the delay bounds.

Subgraph G_1 has 288 events. During Phase I, bounds on the time separation of 129 pairs of events are monitored to check for timing constraint violations and to estimate bounds on performance metrics. Phase I converges after 2 iterations and takes 0.46s on a 150 MHz MIPS R5000 machine with 128 MB of memory. In comparison, the symbolic performance analy-

sis technique of Xie and Beerel [27] takes 16.5 minutes on a SPARC-20 with 64 MB of memory to analyze the performance of the same system. In Phase II, the behavior of DiffEq from Reset to the end of the fourth iteration of the dataflow graph is modeled and analyzed. This graph contains 680 events, and bounds on the time separation of 441 pairs of events are monitored to check for timing constraint violations. This phase of analysis takes 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 may be inexact. Therefore, for comparison, the analysis with $\pm 5\%$ delay variations is repeated with McMillan and Dill's branch-and-bound algorithm [8] (an exact algorithm for acyclic timing constraint graphs) used in place of the AcyclicApproxSep algorithm. All the bounds are found to be identical to those obtained with AcyclicApproxSep. However, the complete analysis (Phases I and II) using McMillan and Dill's algorithm takes 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 II. Each entry lists a pair of events, the required minimum separation between these events, and the computed bounds on 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 III shows the loop delay and cycle time of DiffEq obtained with 0%, $\pm 5\%$ and $\pm 10\%$ variations in delays. The 0% entries show the loop delay and cycle time variation due to data-dependent delay variations. The 5% and 10% entries show the added variation in performance due to small variations in component delays due to manufacturing and operating condition variations.

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]

TABLE III
PERFORMANCE ANALYSIS RESULTS. ALL TIMES ARE IN ns.

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 Fig. 8b. This was motivated by feedback from the designers, who had observed by means of 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 II 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.

VI. DRAWBACKS AND LIMITATIONS

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

- Currently, timing constraint graphs are manually constructed from other descriptions (e.g. RTL) of the system behavior. For large and complex systems such as DiffEq, this is a tedious and error-prone task. Additional research on automating (at least partially) the generation of timing constraint graphs from standard system descriptions needs to be done in order to facilitate the widespread use of the proposed method. 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 is an important feature of many practical asynchronous systems. Since tightly-coupled systems cannot model choice, 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 analyzed 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 the 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 behavior. This may be seen from the proofs of Theorem 1 and Corollary 1, which, in turn, are needed to prove Lemma2 and Theorem 2. Systems satisfying properties **P1** through **P3**, but not **P4**, or systems with infinite edge delay bounds may still be analyzed 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 analyzing such systems with the proposed algorithm is unclear. Note, however, that the computed bounds are conservative regardless of whether property **P4** is satisfied or whether the edges have finite delays.
- 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 some other predecessors. In addition, linear constraints may exist between the time of the current event and those of other events. Unfortunately, systems with multiple types of constraints associated with the same event cannot be modeled or analyzed with the proposed technique.

VII. Conclusion

This paper presented a pseudo-polynomial time algorithm for computing conservative bounds on the time separation of events in a class of choice-free asynchronous systems with min and max type timing constraints. The systems considered are more general than those considered earlier by Amon et al. [16], Hulgaard et al [17] and Myers et al. [15]. Although the algorithm is based on an idea similar to Myers and Meng's algorithm [15], there are several important extensions to and differences from their work. We believe that these represent important contributions to the existing body of work on timing analysis of asynchronous systems with min and max timing constraints. A complete asynchronous differential equation solver chip has been modeled and analyzed using the proposed technique, demonstrating the practicality of the approach. Although the analysis is conservative in the worst-case, our experiments (DiffEq and other examples adapted from the literature [12], [13], [10], not all of which could be described in the paper due to lack of space) yielded exact results. This suggests that carefully designed efficient algorithms for approximate timing analysis of asynchronous systems can be a promising alternative to more expensive exact techniques. Such techniques can be extremely useful in design-analyze-redesign environments, or may even 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 gain wider acceptance in niche application domains.

Acknowledgments

We are thankful to Julio Arceo of Qualcomm Inc., San Diego, CA for helping us with SPICE simulations. We are also thankful to the anonymous reviewers for providing valuable suggestions that greatly helped us improve the quality of the paper.

References

[1] W. Chou, P. A. Beerel, R. Ginosar, R. Kol, C. J. Myers, S. Rotem, K. Stevens, and K. Y. Yun, "Average-case optimized technology mapping of one-hot domino circuits," in *Proceedings of the 4th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1998, pp. 80–91.

[2] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev, "RAPPID: An asynchronous instruction length decoder," in *Proceedings of the 5th International Symposium on Advanced Research in Asynchronous Circuits and Systems (to appear)*, Apr. 1999.

[3] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo, "The design and verification of a high-performance low-control-overhead asynchronous differential equation solver," *IEEE Transactions on VLSI Systems*, vol. 6, no. 4, pp. 643–655, Dec. 1998.

[4] G. Borriello, *A new interface specification methodology and its application to transducer synthesis*, Ph.D. thesis, University of California at Berkeley, 1988.

[5] T. Gahlinger, *Coherence and Satisfiability of Waveform Timing Specifications*, Ph.D. thesis, University of Waterloo, 1990.

[6] J. A. Brzozowski, T. Gahlinger, and F. Mavaddat, "Consistency and satisfiability of waveform timing specifications," *Networks*, vol. 21, no. 1, pp. 91–107, Jan. 1991.

[7] F. Mavaddat and T. Gahlinger, "On deducing timing constraints in the verification of interfaces," *Formal Methods in System Design*, vol. 12, no. 3, pp. 223–239, Apr. 1998.

[8] K. L. McMillan and D. L. Dill, "Algorithms for Interface Timing Verification," in *Proceedings of the IEEE International Conference in Computer Design: VLSI in Computers and Processors*, Oct. 1992, pp. 48–51.

[9] P. Vanbekbergen, G. Goossens, and H. De Man, "Specification and analysis of timing constraints in signal transition graphs," in *Proceedings of the European Design Automation Conference*, Mar. 1992, pp. 302–306.

[10] C. J. Myers, *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*, Ph.D. thesis, Stanford University, 1995.

[11] E. A. Walkup, *Optimization of Linear Max-Plus Systems with Application to Timing Analysis*, Ph.D. thesis, University of Washington, Seattle, 1995.

[12] T. Amon, *Specification, Simulation and Verification of Timing Behavior*, Ph.D. thesis, University of Washington, Seattle, 1993.

[13] H. Hulgaard, *Timing Analysis and Verification of Timed Asynchronous Circuits*, Ph.D. thesis, University of Washington, Seattle, 1995.

[14] T.-Y. Yen, A. Ishii, A. Casavant, and W. Wolf, "Efficient algorithms for interface timing verification," *Formal Methods in System Design*, vol. 12, no. 3, pp. 241–265, Apr. 1998.

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

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

[17] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello, "An algorithm for exact bounds on time separation of events in concurrent systems," *IEEE Transactions on Computers*, vol. 44, no. 11, pp. 1306–1317, Nov. 1995.

[18] T. Amon and G. Borriello, "An approach to symbolic timing verification," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 1992, pp. 410–413.

[19] T. M. Burks and K. A. Sakallah, "Min-max linear programming and the timing analysis of digital circuits," in *Proceedings of the International Conference on Computer-Aided Design*, Nov. 1993, pp. 152–155.

[20] S. M. Burns, *Performance Analysis and Optimization of Asynchronous Circuits*, Ph.D. thesis, California Institute of Technology, 1991.

[21] T. K. Lee, *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits*, Ph.D. thesis, California Institute of Technology, 1995.

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

[23] T. E. Williams, "Performance of iterative computation in self-timed rings," *Journal of VLSI Signal Processing*, vol. 7, no. 1-2, pp. 17–31, Feb. 1994.

[24] C. D. Nielsen and M. Kishinevsky, "Performance analysis based on timing simulation," in *Proceedings of the ACM/IEEE Design Automation Conference*, Oct. 1994, pp. 70–76.

[25] C. Tofts, "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*, Nov. 1995.

[26] P. Kudva, G. Gopalakrishnan, and E. Brunvand, "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.

[27] A. Xie and P. A. Beerel, "Symbolic techniques for performance analysis of timed systems based on average time separation of events," in *Proceedings of the 3rd International Symposium on Advanced Rearch in Asynchronous Circuits and Systems*, Apr. 1997, pp. 64–75.

[28] J. Ebergen and R. Berks, "Response time properties of some asynchronous circuits," in *Proceedings of the 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1997, pp. 76–86.

[29] R. Berks and J. Ebergen, "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*, Dec. 1997, pp. 179–188.

[30] H. Hulgaard and S. M. Burns, "Bounded delay timing analysis of a class of CSP programs with choice," in *Proceedings of the 1st International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Nov. 1994, pp. 2–11.

[31] P. Girodias, E. Cerny, and W. J. Older, "Solving linear, min and max constraint systems using CLP based on relational arithmetic," *Theoretical Computer Science*, vol. 173, no. 1, pp. 253–281, Feb. 1997.

[32] T. Amon, G. Borriello, D. Hu, and J. Liu, "Symbolic timing verification of timing diagrams using Presburger formulas," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 1997, pp. 226–231.

[33] S. Chakraborty, D. L. Dill, K. Y. Yun, and K.-Y. Chang, "Timing analysis for extended burst-mode circuits," in *Proceedings of the Third International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1997, pp. 101–111.

[34] S. Chakraborty, *Polynomial-Time Techniques for Approximate Timing Analysis of Asynchronous Systems*, Ph.D. thesis, Stanford University, Aug. 1998.

[35] K. Y. Yun, *Synthesis of asynchronous controllers for heterogeneous systems*, Ph.D. thesis, Stanford University, 1994.
[36] S. Chakraborty and D. L. Dill, "Approximate algorithms for time separations of events," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Nov. 1997, pp. 190–194.
[37] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo, "The design and verification of a high-performance low-control-overhead asynchronous differential equation solver," in *Proceedings of the 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1997, pp. 140–153.

APPENDIX

Proof of Theorem 1:
Proof: Let there be k events in each of C_1 and C_2. Without loss of generality, the events in C_1 are assigned topological indices 0 through $k-1$, and those in C_2 are assigned topological indices in $n-k$ through $n-1$. The theorem is proved by showing that $\Delta(i,j)$ is bounded above by $L(j)-l(i)$ for all i and j in $n-k$ through $n-1$.

Let i be an integer in the above range. The proof consists of three parts. First, we prove the bound for $\Delta(i,j)$ when j lies in 0 through $k-1$. Next, we consider $\Delta(i,j)$ with j in the range k through $i-1$. Finally, we consider $\Delta(i,j)$ when j lies in $i+1$ through $n-1$. Note that in the first two cases, i is greater than j, so $\Delta(i,j)$ is computed in the i^{th} iteration of the outermost for loop of algorithm AcyclicApproxSep (see Fig. 3). However, in the third case, j is greater than i, so $\Delta(i,j)$ is computed in the j^{th} iteration of the outermost loop.

Part I: By property **P4** in Definition 1, there exists a path P from event j to event i such that all events along P have max type constraints. Let v be the predecessor of i along P. It follows from step (i) of function ComputeInitialEstimates (see Fig. 3) that

$$\Delta(i,j) \leq \Delta(v,j) - d_{v,i}. \tag{1}$$

However, v itself lies on path P, so it is a max type vertex, and index j is less than index v. Let u be the predecessor of v along P. From step (i) of function ComputeInitialEstimates, we again have

$$\Delta(v,j) \leq \Delta(u,j) - d_{u,v}. \tag{2}$$

Continuing this argument and adding the resulting inequalities, we obtain

$$\begin{aligned}\Delta(i,j) &\leq \Delta(j,j) - \sum_{\langle x,y\rangle \text{ along } P} d_{x,y} \\ &\leq -l(i) &(3)\\ &\leq L(j) - l(i). &(4)\end{aligned}$$

Inequality (3) 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 (4) follows from the observation that j is in C_1, so $L(j)$ is at least 0 by definition.

Part II: Now, consider $\Delta(i,j)$, with j lying in the range k through $i-1$. The proof is by complete induction on j.
Basis: From part **I**, it follows that for all j in 0 through $k-1$, the entry $\Delta(i,j)$ is bounded above by $L(j)-l(i)$.
Hypothesis: Let j be an integer in k through $i-1$. For all r in 0 through $j-1$, let $\Delta(i,r)$ be bounded above by $L(r)-l(i)$.
Induction: It follows from steps (2a) and (2c) of algorithm AcyclicApproxSep that

$$\begin{aligned}\Delta(i,j) &\leq \max_{k \in preds(j)} (\Delta(i,k) + D_{k,j}) \\ &\leq \max_{k \in preds(j)} (L(k) - l(i) + D_{k,j}) &(5)\\ &\leq L(j) - l(i). &(6)\end{aligned}$$

Inequality (5) follows from the hypothesis, and inequality (6) 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 $\Delta(i,j)$, with j lying in the range $i+1$ through $n-1$. The proof is by complete induction on j.
Basis: Follows from parts **(I)** and **(II)**.
Hypothesis: Let j be an integer in the range $i+1$ through $n-1$. For all r in 0 through $j-1$, let $\Delta(i,r)$ be bounded above by $L(r)-l(i)$.
Induction: Since $\Delta(i,j)$ is computed in iteration j, it follows from steps (ii) and (iv) of function ComputeInitialEstimates that

$$\Delta(i,j) \leq \max_{k \in preds(j)} (\Delta(i,k) + D_{k,j}) \tag{7}$$

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 $L(j)-l(i)$.

■

Proof of Lemma 1:
Proof: We start with the following observation about algorithm AcyclciApproxSep.

Observation 1 *After initialization, the only steps in which elements of the Δ matrix are updated are steps (i) through (iv) in function* ComputeInitialEstimates *and steps (2a) through (2d) in function* AcyclicApproxSep. *The expressions on the right hand side of the assignments in each of these steps involve only non-negated Δ matrix entries. Therefore, if all edge delay bounds are kept unchanged and the bounds between every pair of source events are either decreased or kept unchanged, the bounds computed by algorithm* AcyclicApproxSep *must decrease or remain unchanged.*

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 1 that $\Delta_2 \leq_{\mathrm{M}} \Delta_1$.
Hypothesis: For i greater than or equal to 1, let $\Delta_{i+1} \leq_{\mathrm{M}} \Delta_i$.
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 Fig. 4), $\Delta_{i+1}(a_2,b_2)$ 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 1. ■

Proof of Lemma 2:

Proof: The constant term, 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, $\Delta(a,b)$ is bounded below by $-\Delta(b,a)$.

From Theorem 1, we know that for every pair of distinct events u and v in C_2, the value of $\Delta(u,v)$ is bounded above by $L(v) - l(u)$ 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. ■

Proof of Theorem 2:

Proof: If at least two iterations are made, Corollary 1 and step 2(b)(iii) of Phase I of algorithm CyclicApproxSep (see Fig. 4) imply that the bounds between every pair of source events in C_1 are finite in the final iteration. Since all edge delay bounds are also finite, it is easy to see from the steps in algorithm AcyclicApproxSep (see Fig. 3) that all Δ matrix entries computed by the algorithm in the final iteration are finite.

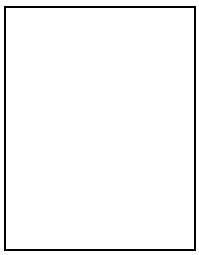
The remainder of the theorem is proved by showing that if the analysis is allowed to continue for i iterations where $i > K$, then the upper bounds on time separations computed in iteration i apply to the events in subgraph G_i. Since $\Delta_i \leq_{\mathrm{M}} \Delta_K$ for all $i \geq K$ (by Lemma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

An important property of algorithm AcyclicApproxSep, proved in [34], and used here, is that given upper bounds on the time separation between all pairs of source events, the bounds computed by AcyclicApproxSep are always at least as large as the corresponding exact upper bounds.

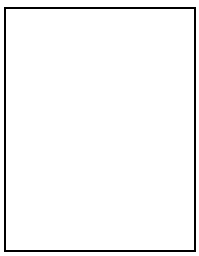
Basis $(i = 1)$: The first iteration is viewed as analysis of G_1. Note that the source events of G_1 belong to cutset C_1. Since $\Delta_1(a_1,b_1)$ is set to $+\infty$ for every a_1 and b_1 in C_1, the value of $\Delta_1(a_1,b_1)$ is an upper bound on the actual time separation from event a_1 to event b_1. Therefore, the value of $\Delta(u_1,v_1)$ computed by algorithm AcyclicApproxSep is an upper bound on 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 $i \geq 2$.

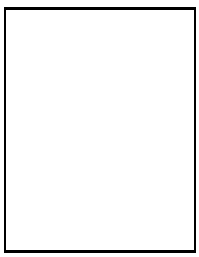
Induction: The i^{th} iteration is viewed as analysis of G_i. The source events of G_i belong to cutset C_i. By hypothesis, the separation from a_i to b_i, for every pair of events a_i and b_i in C_i, is bounded above by $\Delta_{i-1}(a_2,b_2)$. Since the value of $\Delta_{i-1}(a_2,b_2)$ is copied to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



Supratik Chakraborty (Member, IEEE) received the B. Tech. degree in Computer Science and Engineering from Indian Institute of Technology, Kharagpur in 1993, and the M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA in 1995 and 1998, respectively. He is currently a member of the Advanced CAD Research group at Fujitsu Laboratories of America, Inc., Sunnyvale, CA. His research interests include the design, analysis, optimization and verification of digital systems, with emphasis on asynchronous systems and fast timing analysis algorithms. Dr. Chakraborty has received the President of India Gold Medal for best academic performance at Indian Institute of Technology, Kharagpur in 1993. He has also received a Best Paper Award at the International Conference on Computer Design in 1998.



Kenneth Y. Yun (Member, IEEE) received the Ph.D. degree in Electrical Engineering from Stanford University, Stanford, CA, and the S.M. in Electrical Engineering and Computer Science from Massachusetts Institute of Technology (MIT), Cambridge. He is currently an assistant professor in the Dept. of Electrical and Computer Engineering at University of California, San Diego. He had held design engineering positions at TRW and Hitachi for 6 years. His current research interests include the design, synthesis, analysis, and verification of mixed-timed VLSI circuits and systems: in particular, interface design methodologies and tools to facilitate ultra-high-speed communications between synchronous/asynchronous modules. He has been working with Intel Corp. as a primary consultant on the Asynchronous Instruction Decoder Project. Dr. Yun has organized ASYNC'98 as a Program Co-Chair. He is a recipient of 1996-99 National Science Foundation CAREER award and 1996 Hellman Faculty Fellowship, and he is a co-recipient of the Charles E. Molnar award for a paper that best bridges theory and practice of asynchronous circuits and systems at ASYNC'97. He has also received a Best Paper Award at the International Conference on Computer Design in 1998.



David L. Dill (Member, IEEE) received the S.B. degree in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology (MIT), Cambridge in 1979 and the M.S. and Ph.D. from Carnegie-Mellon University, Pittsburgh, PA, in 1982 and 1987, respectively. He is an Associate Professor of Computer Science and (by courtesy) Electrical Engineering at Stanford University, Stanford, CA. He has been on the faculty at Stanford since 1987. His primary research interests relate to the theory and application of formal verification techniques to system designs, including hardware, protocols, and software. Dr. Dill's Ph.D. dissertation, "Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuits" was named as a Distinguished Dissertation by ACM and published as such by M.I.T. Press in 1988. He was the recipient of an Presidential Young Investigator award from the National Science Foundation in 1988, and a Young Investigator award from the Office of Naval Research in 1991. He has received Best Paper awards at International Conference on Computer Design in 1991 and the Design Automation Conference in 1993 and 1998. From July 1996 to September 1997 he was Chief Scientist of 0-In Design Automation.