Practical Timing Analysis of Asynchronous Systems Using Time Separation of Events *

Supratik Chakraborty †
supratik@celery.stanford.edu

Kenneth Y. Yun ‡
kyy@paradise.ucsd.edu

David L. Dill †
dill@cs.stanford.edu

† Computer Systems Laboratory, Stanford University, Stanford, CA 94305
‡ Dept. of Electrical and Computer Engg., University of California, San Diego, CA 92093-0407

Abstract

We present a unified technique for timing verification and performance analysis of complex asynchronous systems designed with implicit timing assumptions. Our method models interacting controllers and datapath elements using timing constraint graphs. Performance metrics and timing constraints to be verified are formulated as time separations between appropriate events. Separations between all relevant pairs of events are computed in a single pass using an efficient time separation of events algorithm [2]. A complete asynchronous differential equation solver chip [16] has been modeled and analyzed using the techniques presented in this paper. We present results of datapath timing verification, inter-controller protocol timing verification and performance analysis of the differential equation solver using our techniques.

1. Introduction

There is mounting evidence that asynchronous systems are finding niches in high-performance applications, such as Intel's asynchronous instruction length decoder and the asynchronous differential equation solver benchmark circuit [16]. 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 the correctness of these highly concurrent systems as well as to estimate their performance, efficient and accurate timing analysis techniques are necessary.

*This work was partially supported by a grant from the Semiconductor Research Corporation (Contract No. 96-DJ-389) and a gift from Sun Microsystems.

Previous work in this area [1, 13, 9, 11, 4, 14, 16] has typically addressed the performance analysis and timing verification problems *separately*. Timing verification of asynchronous systems has been primarily concerned with controller timing and inter-controller protocol verification. Asynchronous datapaths have traditionally been considered free of timing problems by virtue of their design style. However, aggressive design styles used in recent high-performance asynchronous systems (e.g., [16]) make implicit timing assumptions in the datapath design as well. Consequently, timing constraints in *both* the datapath *and* the control need to be checked in order to ensure correct operation of these systems.

In this paper, we propose a *unified analysis technique* for performance analysis and timing verification (both controller and datapath) of complex asynchronous systems designed with implicit timing assumptions. Our method proceeds by modeling interacting asynchronous controllers and datapath elements using *timing constraint graphs*. Performance metrics and timing constraints to be verified are formulated as time separations between appropriate events in the timing constraint graph. Separations between *all* relevant pairs of events are then computed in *a single pass* using an efficient time separation of events algorithm [2]. This effectively solves both the performance analysis and timing verification problems simultaneously. We also present an interesting analytical result showing that the steady-state behavior of a system of communicating controllers can be analyzed without any knowledge about the past behavior of the system. This greatly simplifies the analysis of the temporal behavior of concurrent systems. The techniques described in this paper have been applied to model and analyze a high-performance asynchronous differential equation solver [16]. This illustrates the feasibility of our approach

for analyzing complex real-life asynchronous systems.

1.1. Related work

The first successful effort at modeling the timing behavior of relatively complex asynchronous systems was by Burns [1]. Lee [7] subsequently extended this work to model inherently disjunctive systems. Their works provide powerful analytical means for estimating and optimizing the performance of asynchronous systems described by handshaking expansions or production rules. Several other performance analysis techniques for special classes of asynchronous systems operating under specific delay models have also been described by other researchers [13, 10, 11, 4, 6, 14].

Timing verification of control-oriented circuits using timed state space search was used in [9]. An improved technique for analyzing the reachable timed state space of systems modeled as labeled safe Petri Nets was described in [12]. [3] describes a simulation-based technique for timing verification of extended burst-mode circuits. In [16], a technique for finding timing bugs in an asynchronous differential equation solver design has been reported.

The work that comes closest to ours is that of Hulgaard [5]. He considered cyclic systems with conjunctive causality and bounded delays, and proposed an algebraic technique for computing the exact separation between two specific events in an infinite unfolding of the system. An extension of this technique to analyze safe Petri Nets with choices was also proposed in [5]. As in our approach, Hulgaard's work can be used for both timing verification and performance analysis of asynchronous systems. However, the algorithms in [5] compute the separation between one pair of events at a time; moreover, this computation has a *worst-case* complexity exponential in the system size. While it has been shown that some reasonably large systems can be analyzed with the algorithms proposed in [5], the *worst-case* exponential complexity of the algorithms can indeed pose serious practical problems when analyzing large and complex systems in general. In addition, modeling min-constraints or OR-causality is cumbersome using Hulgaard's approach. Nevertheless, the work of [5] represents an important contribution to the available techniques for timing analysis of asynchronous systems.

2. A motivating example

This section provides an overview of a high-performance asynchronous differential equation solver (subsequently referred to as Diffeq) that serves as a motivating example for our modeling and analysis techniques. Fig. 1, adapted from [16], shows the dataflow graph and the architecture of Diffeq. Details of the design can be found in [16].

The control logic of Diffeq is implemented using four *distributed* extended burst-mode FSMs [15] (ALU1 control, ALU2 control, MUL1 control and MUL2 control in Fig. 1b). These controllers communicate with one another using a protocol which assumes safe timing bounds in signaling, unlike *speed-independent* or *delay-insensitive* designs. The datapath is structurally identical to synchronous designs, except for the completion-reporting logic associated with datapath elements. Like synchronous systems, the control is responsible for generating ALU opcodes, MUX selects, register load enables, and precharge/evaluate signals. However, there are implicit assumptions on the timing of datapath signals, which must be satisfied for correct functioning of the system. For example, data inputs to domino circuits must stabilize before an evaluation begins, and setup and hold-time constraints of registers must be satisfied.

3. Modeling

We now describe techniques for modeling systems of communicating asynchronous controllers and datapath elements with timing constraint graphs. We limit our discussion to systems with extended burst-mode controllers. However, our techniques can easily be extended to model other types of asynchronous controllers as well.

3.1. Timing constraint graphs

We use timing constraint graphs similar to those described in [8]. Vertices in the graph represent events and directed edges represent causal dependencies between events. A directed edge from event i to event j is labeled with an interval $[d_{i,j}, D_{i,j}]$ representing the bounded delay in the propagation of event i to event j. Each non-source vertex i (having at least one incoming edge) is also labeled with a min or max operation, which determines how the time of occurrence of i depends on its immediate predecessors in the graph. For example, if there exist directed edges from vertices j and k to vertex i and if vertex i has a min label, then: $t_i = \min(t_j + [d_{j,i}, D_{j,i}],\ t_k + [d_{k,i}, D_{k,i}])$, where t_x represents the time of occurrence of event x.

3.2. Modeling controller timing

Our approach to modeling the temporal behavior of a system of interacting controllers consists of two basic steps:

1. Model the timing behavior of each controller separately by analyzing its state transition diagram.

2. "Stitch" the timing constraint graphs of the individual controllers by considering inter-controller synchro-

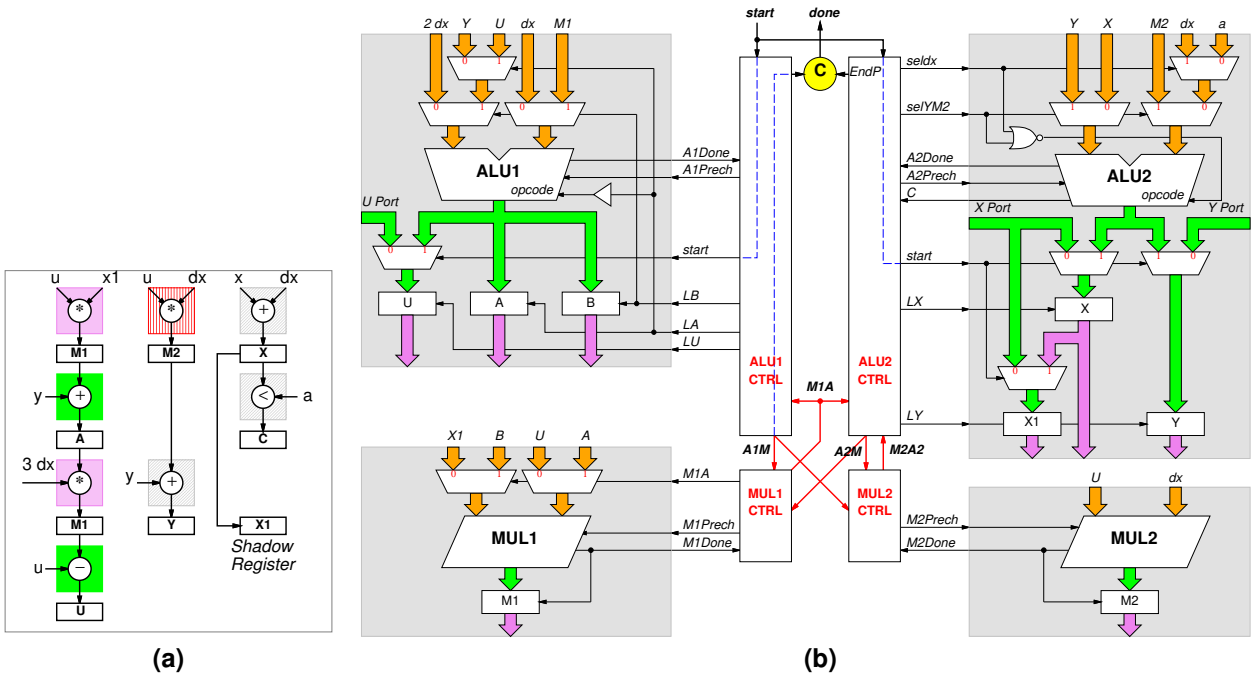

Figure 1. (a) Dataflow graph and (b) Architecture of differential equation solver

nization and the interaction of the controllers with datapath components.

This section describes these steps in detail, using the controllers of Diffeq for illustration.

For purposes of modeling, we view each individual 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. The precise timing constraints between the input and output transitions depend on the type of controller and also on internal circuit delays. For example, an extended burst-mode controller will assert its outputs only after all transitions in an input burst have occurred. Depending on the implementation of the controller, the delay from the last transition in an input burst to an enabled output transition may also vary widely. In this paper, we choose not to address internal controller timing issues for clarity of exposition. Thus, we will assume that are given the type of the controller (e.g., extended burst-mode) and also the delays from the input transitions to the output transitions triggered by them. This black-box model of the controller suffices for purposes of performance analysis and datapath and inter-controller protocol timing verification. Note, however, that the techniques presented in this paper can be easily extended to model internal gates inside a controller and verify internal controller timing as well.

We restrict the following discussion to extended burst-mode controllers. For each controller, we analyze its state transition diagram to obtain constraints between the transitions in an input burst and the output transitions enabled by the burst. For each state transition, let $\mathcal{I}$ be the set of input transitions and $\mathcal{O}$ the set of output transitions. Extended burst-mode semantics then require that *all terminating* transitions [15] 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. This behavior is modeled in a timing constraint graph as shown in Fig. 2b. We create a max-type vertex, m, and draw zero-delay edges from the terminating transitions in $\mathcal{I}$ to m. We also draw edges from m to each concurrently enabled transition in $\mathcal{O}$, and label each such edge with the delay incurred by the controller circuit in generating the corresponding output transition after the last terminating transition in $\mathcal{I}$ has occurred. This process is then repeated for each state transition in the state transition diagram.

Input events, such as M2done− and A2M− in Fig. 2b, are generated either by datapath components or by some other controllers as part of the inter-controller communication protocol. For example, M2done− in Fig. 2b is generated by the MUL2 multiplier of Fig. 1b in response to M2Prech+ generated by the MUL2 controller. The signal M2prech+ initiates the precharging of the MUL2 multiplier, and M2done− notifies the MUL2 controller that all the internal nodes of the multiplier have been precharged. If the precharging delay varies between 4 and 5 units of time,

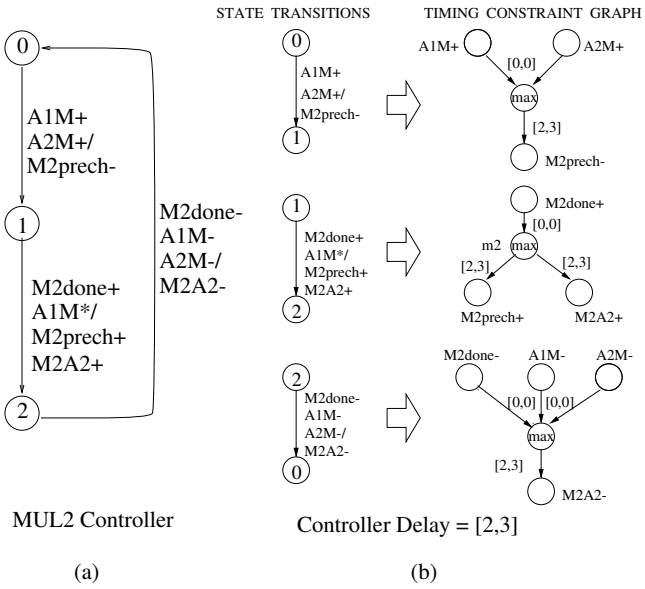

Figure 2. Modeling the controller behavior.

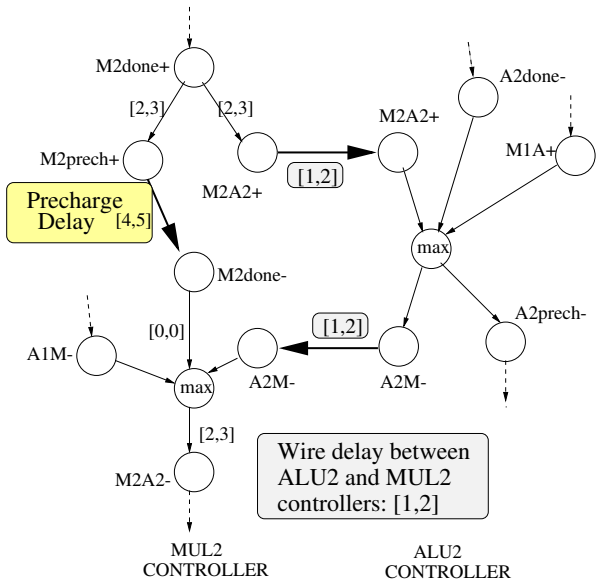

Figure 3. "Stitching" timing constraint graphs.

we can draw an edge from M2prech+ to M2done− and label it $[4, 5]$, as shown in Fig. 3. Unlike M2done−, however, A2M− represents a synchronization event generated by the ALU2 controller and communicated to the MUL2 controller. This event, therefore, appears as an output event in the timing constraint graph for the ALU2 controller, and as an input event in the graph for MUL2 controller. These two events (vertices) may now be connected by a directed edge labeled with the wire delay from the ALU2 controller to the MUL2 controller. Continuing this process, the timing constraint graphs of the individual controllers can be "stitched" together using synchronization events. Fig. 3 shows fragments of the timing constraint graphs for the MUL2 and ALU2 controllers linked at M2A2+ and A2M−.

3.3. Modeling datapath timing

Unlike controllers, the timing behavior of datapath components vary significantly from one type of component to another. In this section, we describe the modeling of three important datapath components. The behavior of other components can be modeled using similar techniques.

3.3.1. Multiplexers

Fig. 4a shows a standard 2-to-1 multiplexer. Depending on the value of the Control input, either DataIn0 or DataIn1 is routed to the DataOut port. If there is a transition on the Control input, then the data at the output port is not valid until Control has stabilized. Similarly, if a data from a particular input port is to be routed to the output port (e.g., if the control is stable 0 or stable 1), the data at the output port is invalid until the corresponding data input has stabilized to its final value. This behavior is modeled using the timing constraint graphs of Figure 4b. The delay from the Control input to the DataOut port is measured assuming that the DataIn ports are stable. Similarly, the delay from the DataIn0 (DataIn1) port to the DataOut port is measured assuming that the Control is stable at 0 (1). Using these delay values in Figure 4b gives a reasonable model of the true behavior of the multiplexer. Note that this technique can be easily extended to model larger multiplexers (e.g., 4-to-1 multiplexers).

3.3.2. Arithmetic circuits designed in domino logic

A typical domino-logic arithmetic circuit is shown in Fig. 5a. The Done output is used in asynchronous systems to indicate the completion of computation. The timing behavior of the circuit is depicted in Fig. 5b. When the precharge signal is asserted (Precharge+), the internal nodes of the circuit are pulled up to V_{DD} after some precharge delay. This drives the outputs of the domino circuit low (DataOutLow or Done−). The precharge signal is then de-asserted and the evaluate signal asserted. Assuming that the data inputs are stable during the evaluation phase, the domino-logic now computes its function and asserts the outputs (DataOutValid or Done+) after some data-dependent computation-delay. After the data outputs have been asserted, the Evaluate signal is de-asserted and

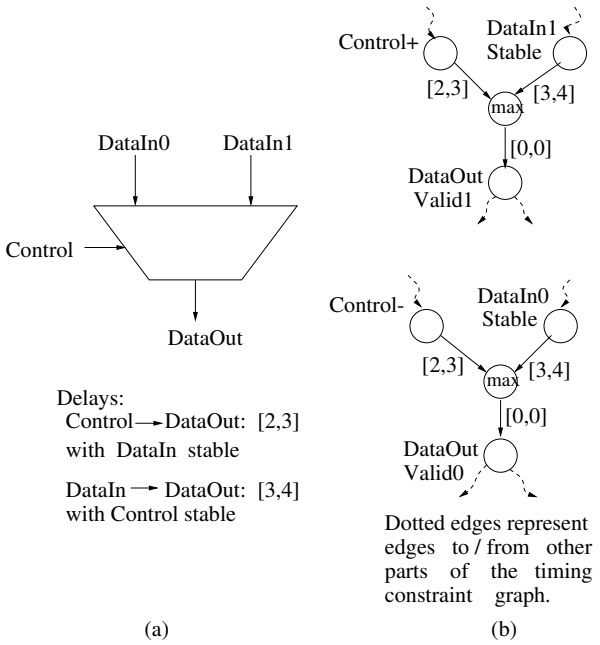

Figure 4. Modeling a 2-to-1 multiplexer

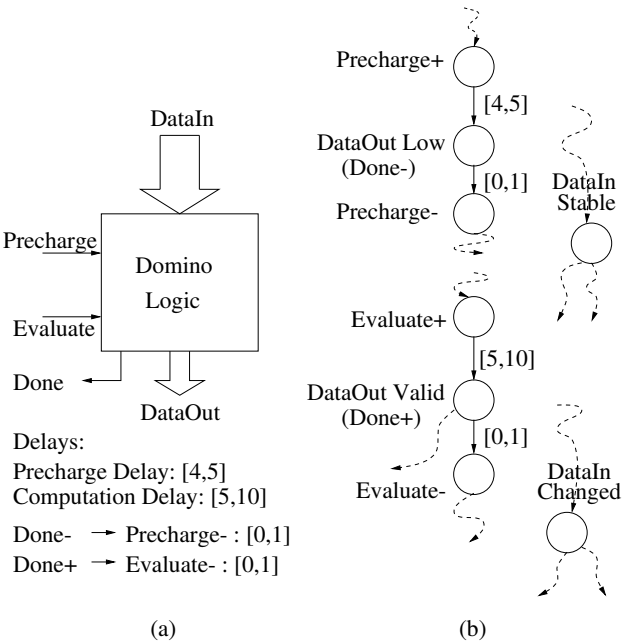

Figure 5. Modeling domino-logic circuits

one precharge-evaluation cycle is completed. The events DataInStable and DataInChanged in Fig. 5b represent the stabilization of the data inputs prior to the evaluation phase and their subsequent change after the evaluation phase.

3.3.3. Edge-triggered registers

Edge-triggered registers can be modeled using techniques similar to those described above. As a simple example, let us consider a positive edge triggered register without asynchronous set/reset inputs (Fig. 6a). Assuming that the data inputs are stable within a setup and hold-time window around the rising edge of the clock input, the value at the data input port is loaded into the register by the rising edge of the clock, and appears at the data-out port after the "clock-to-q" delay of the register. We model this as shown in Fig. 6b. As in the model of the domino circuit, DataInStable and DataInChanged represent the stabilization of the data inputs prior to latching and their subsequent change.

4. Uniform problem formulation

In this section, we formulate the performance metrics and timing constraints as time separations between appropriate events in the system. This allows us to use an efficient time separation of events algorithm [2] to solve both the performance analysis and timing verification problems

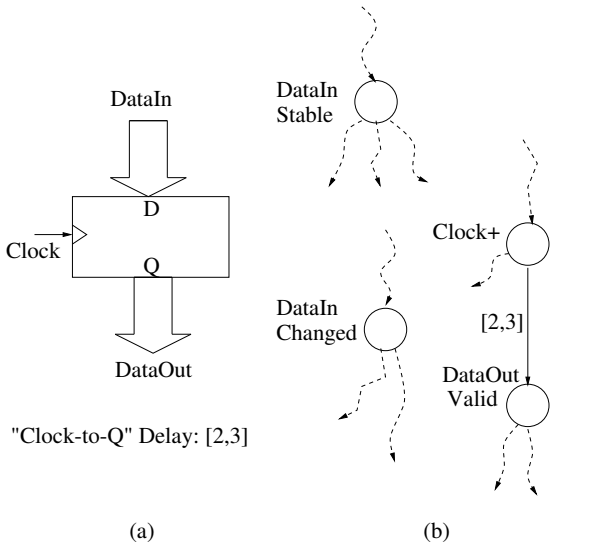

Figure 6. Modeling an edge-triggered register

simultaneously. We use the Diffeq example to illustrate our approach.

4.1. Formulating performance metrics

We focus on two important performance metrics of the Diffeq example: (i) the *latency* of one iteration of the dataflow graph (see Fig. 1a), and (ii) the *cycle-time* as Diffeq executes successive iterations (cycles) of the dataflow graph.

To compute the latency, we consider the timing constraint graph of Diffeq for one iteration of the dataflow graph. Since there are multiple parallel threads of computation, we represent the starting time of an iteration by taking the minimum start times of all the threads. A min-type vertex, StartOfIter, is thus added to the timing constraint graph. Similarly, the end of computation in an iteration is represented by taking the maximum. In this case, a max-type vertex, EndOfIter, is added to the timing constraint graph. The time separation between EndOfIter and StartOfIter then gives the latency of one iteration of the dataflow graph.

Estimating the cycle-time involves determining the separation between two similar events in successive iterations of the dataflow graph. We have seen above how the start of the current iteration can be modeled using a min-type event. The start of the next iteration can be modeled similarly by taking the minimum start times of the *next* iteration. For each thread, this occurs immediately after it finishes computing the current iteration. We, therefore, add another min-type vertex, called StartOfNextIter, in the timing constraint graph to represent this. Computing the time separation between StartofIter and StartOfNextIter then yields the cycle-time of the system.

4.2. Formulating timing constraints for verification

Timing constraints in the Diffeq datapath consist of setup and hold-time constraints of registers and setup time constraints of domino logic circuits. In order to check register timing constraints, we analyze each instance of loading data into a register. For each such instance, we determine the minimum separation between the events DataInStable and Clock+ shown in Fig. 5b. If this separation exceeds the setup-time requirement of the register, we know that there is no setup-time violation. Similarly, checking register hold-time constraints amounts to determining whether the minimum separation between Clock+ and DataInChanged is greater than the hold-time requirement of the register.

In order to verify the timing of domino-logic circuits, we consider each instance of the domino circuit performing some computation. For each such instance, we determine the minimum separation between the stabilization of inputs and the arrival of the evaluate signal. A negative value implies that the input data can potentially change after the evaluate signal has arrived. This can lead to accidental discharge of the internal nodes, resulting in incorrect results. Note that if the input data lines are guaranteed to rise monotonically, the above timing requirement is not necessary for the correct functioning of the domino circuits. However, the monotonic rising constraint is often too restrictive; hence ensuring that the data inputs stabilize before the evaluation phase is a robust way of ensuring the correct operation of the circuit.

Finally, we consider verifying the timing constraints of the inter-controller communication protocol. As an example, consider the state transition diagram of the MUL2 controller shown in Fig. 2a. Here, we have a rising transition on A2M enabling the $S_0 \rightarrow S_1$ state transition. Subsequently, A2M is required to remain stable as M2done rises and the controller changes state from S_1 to S_2. Only after this, should A2M fall, enabling the $S_2 \rightarrow S_0$ state transition. This specification gives rise to the constraint that A2M, once it rises, must not fall until M2done rises. In other words, the minimum separation between M2done+ and A2M− must be positive. All other timing constraints in the inter-controller communication protocol of Diffeq are of similar nature, i.e., they seek to ensure that some events always occur after some other events. Such constraints can always be reformulated in terms of time separations between events.

5. Analysis

This section gives an overview of an efficient algorithm for computing bounds on the time separations between all pairs of events in acyclic timing constraint graphs. We also describe an interesting analytical result that enables us to study the steady-state behavior of systems of communicating asynchronous controllers, without any knowledge of the past behavior of the system. This effectively enables us to analyze the temporal behavior of an important class of cyclic systems using timing analysis algorithms for acyclic systems.

5.1. Time separation of events analysis

We compute bounds on the time separations between events using the algorithm of [2]. This algorithm analyzes *acyclic* timing constraint graphs with *both* min and max type constraints, and computes bounds on the time separations between *all* pairs of events in *one pass*. The bounds computed by the algorithm are *conservative* in the worst-case; however, experiments on several large benchmarks [2] have shown that the computed bounds are highly accurate and often exact in practice. The computational complexity

of the algorithm is $O(n^2.p)$, where n = number of events and p = maximum fanin of a vertex in the timing constraint graph.

The basic idea behind the algorithm is as follows. A timing constraint graph represents a system of constraints among times of occurrences of events. Some of these constraints are non-linear, e.g., min and max constraints. The feasible space of such a system of constraints involving non-linear functions is non-convex in general. Consequently, standard convex optimization techniques cannot be applied to find bounds on the time separations between events. The algorithm of [2], therefore, computes a small convex "envelope" that completely contains the original feasible space, and then computes bounds on the differences between the times of occurrences of events in this approximate convex space. Since all points in the original feasible space are contained within the approximate convex space, the computed bounds are conservative, i.e., upper bounds are never under-estimated and lower bounds are never over-estimated. In practice, however, the bounds computed by the algorithm of [2] have a very high degree of accuracy. Further details about the algorithm may be found in [2].

5.2. Practical systems of interacting controllers

Most practical systems of interacting controllers are "tightly-coupled" in the sense that each controller synchronizes with *all other* controllers, directly or indirectly, between two successive iterations through its state transition loop. In this Section, we describe how the steady-state temporal behavior of such tightly-coupled systems can be analyzed without any knowledge of the past behavior of the system. This also gives us a means of performing timing verification of tightly-coupled cyclic systems using the acyclic algorithm of [2].

We first construct the timing constraint graph of each controller considering *two* complete iterations of its state transition loop. The sink events in the graph (events with no outgoing edges) represent the same signal transitions as the source events (events with no incoming edges), although they occur two complete iterations later. The timing constraint graphs for the individual controllers are linked to each other using the synchronization events, as illustrated in Section 3.2 (Fig. 3). Let the source events of the resulting ensemble of timing constraint graphs be $s_1, s_2, \cdots, s_N$ and let the sink events be $r_1, r_2, \cdots, r_N$. The tight-coupling property and two unfoldings of the state transition loops ensure that none of the events $r_1, \cdots, r_N$ can occur until all the events $s_1, \cdots, s_N$ have occurred. Moreover, if all edges in the timing constraint graphs have finite bounded delays, the system has the following interesting property.

Property P1: The time separation between any two sink events r_k and r_l is finitely bounded regardless of the separations (finite or infinite) between the source events $s_1, \cdots, s_N$.

To prove this, we define the delay of a path from s_i to r_j as the sum of all edge delays along that path. Let $d_{i,j}$ and $D_{i,j}$ be the minimum and maximum of all such path delays from s_i to r_j. Since all edge delays are finitely bounded, $d_{i,j}$ and $D_{i,j}$ are finite. We now observe that if event r_k is triggered by source event s_x, then $(t_{r_k} - t_{s_x}) \leq D_{x,k}$. Because of the tight coupling property, we must also have $(t_{r_l} - t_{s_x}) \geq d_{x,l}$. Combining these two inequalities gives $(t_{r_l} - t_{r_k}) \geq d_{x,l} - D_{x,k}$. Since s_x could be any one of $s_1, \cdots, s_N$, a lower bound of $(t_{r_l} - t_{r_k})$ is $\min(d_{1,l} - D_{1,k}, \cdots, d_{N,l} - D_{N,k})$. Similarly, it can be shown that $\max(D_{1,l} - d_{1,k}, \cdots, D_{N,l} - d_{N,k})$ is an upper bound of $(t_{r_l} - t_{r_k})$. Since all the constants involved are finite, we have effectively proved Property **P1**.

This suggests the following procedure for analyzing the steady-state behavior of tightly coupled systems. We construct timing constraint graphs as before, and perform a first pass of the time separations analysis, assuming that the initial separation between every pair of source events is $[-\infty, +\infty]$. Property **P1** then implies that the bounds on the separations between the sink events, computed by a reasonably accurate and provably conservative algorithm like [2], will be finite. We now feed these computed bounds back as estimates of the initial separations between the source events and repeat the analysis.

Since the initial separations used in the second pass are finitely bounded and, therefore, *completely contained* within the initial separations used in the first pass ($[-\infty, +\infty]$), it follows that the separations between sink events computed in the second pass must be *contained* within the corresponding separations computed in the first pass. Continuing this argument, we find that the separation intervals between sink events monotonically shrink toward a steady value with each pass of analysis. The convergence criterion of our analysis is that the separation between every pair of sink events is exactly equal to (or within a desired margin from) the corresponding separations between the source events. Applying this procedure to the Diffeq design results in convergence in just two passes of analysis.

Notice that we considered 2 iterations through the state transition loop of each controller during each pass of analysis. Consequently, if convergence is detected in the k^{th} pass of analysis, the converged bounds may be regarded as valid only after each controller has cycled through its state transition loop $2(k-1)$ times. Therefore, for timing verification purposes, we must analyze the initial behavior of the system besides performing the steady-state analysis. Specifically, we must analyze the timing constraint graph of the complete system as each controller cycles through its state transition loop $2(k-1)$ times after power up. Note that this timing constraint graph is *acyclic* and therefore amenable

to the analysis algorithm of [2]. For the Diffeq example, we have $k = 2$. Therefore, we need to analyze the first two iterations of the dataflow graph (see Fig 1a) after power up, in addition to performing the steady-state analysis.

Since the state transition loop of each controller is unfolded twice when constructing the ensemble of timing constraint graphs for the steady-state analysis, the graph generated from each individual controller is acyclic. Linking the acyclic timing constraint graphs of individual controllers at synchronization events cannot produce cycles. Otherwise, we would be able to find an event that depends on its successor in some controller's timing constraint graph, for it to occur. However, since the timing constraint graph generated from each controller is acyclic, this cannot happen. The same reasoning also implies that linking the timing constraint graphs of the controllers with those of the datapath components with which they interact cannot produce any cycles. Consequently, the timing constraint graph resulting from linking the individual graphs is *acyclic*. This enables us to use the algorithm of [2] for efficiently computing bounds on the time separations between every pair of events.

6. Experimental Results

We have modeled and analyzed the complete asynchronous differential equation solver of Section 2 using the techniques presented in this paper. The timing constraint graph modeling two iterations of the dataflow graph of Fig. 1a contains 258 events. Delays of the circuit components were obtained from SPICE simulations under typical-case conditions (22°C, 3.3V and typical-case parasitics). We introduced a percentage variation on the SPICE delays in order to obtain the delay bounds in the timing constraint graph. This allows us to observe how percentage variations of the delays affect the circuit's correctness and performance.

Table 1 shows the latency and cycle time of the differential equation solver obtained with $\pm 5\%$ and $\pm 10\%$ variations in delays. We present a subset of our timing verification results in Table 2. Each entry lists a pair of events, the required minimum separation between these events, and the computed bounds on the separation between these events. All times are in ns. Note that the computed bounds are conservative in the worst-case [2]. This implies that if our analyzer finds that a certain timing constraint is satisfied, it is *guaranteed* to be satisfied. However, there may be situations in which our analysis identifies a timing constraint violation, while, in reality, the system is perfectly fine.

The entire analysis (computation of all pairs of event separations) takes less than 0.2 second on an UltraSPARC-1 (143 MHz) with 128 MB of memory. In comparison, the symbolic performance analysis method of [14] takes 16.5 minutes on a SPARC-20 with 64 MB of memory for performance analysis of the same design.

Latency	$\pm 5\%$ variation	[35.5870, 54.8060]
	$\pm 10\%$ variation	[33.7140, 59.3944]
Cycle Time	$\pm 5\%$ variation	[33.4590, 48.7790]
	$\pm 10\%$ variation	[31.6980, 54.1580]

Table 1. Performance analysis for typical-case SPICE delays.

7. Conclusion

In this paper, we have presented a unified approach for timing verification and performance analysis of asynchronous systems. This contrasts with most previous approaches which address these two problems separately. For practical systems of asynchronous controllers, we have also shown that the steady-state temporal behavior can be be analyzed without any knowledge of the past behavior of the system. Finally, we have demonstrated the practicality and feasibility of our approach by actually modeling and analyzing a fabricated asynchronous differential equation solver chip. We believe that efficient timing verification and performance analysis techniques, as the one reported in this paper, are indispensable for the design of timing-critical high-performance asynchronous systems.

7.1. Acknowledgment

We are thankful to Julio Arceo of the University of California, San Diego for helping us with the SPICE simulations.

References

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

[2] S. Chakraborty and D. L. Dill. Approximate algorithms for time separations of events. In *Proceedings of the ICCAD*, Nov. 1997.

[3] S. Chakraborty, D. L. Dill, K. Y. Yun, and K.-Y. Chang. Timing analysis for extended burst-mode circuits. In *Proceedings of the 3rd ASYNC*, April 1997.

[4] J. Ebergen and R. Berks. Response time properties of some asynchronous circuits. In *Proceedings of the 3rd ASYNC*, April 1997.

	Event1 (e1)	Event2 (e2)	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 (SU)	[1.1875, 1.3125]	OK if SU $<$ 1.1875
	LA+	A1done−	Hold-time (HT)	[3.5135, 4.1465]	OK if HT $<$ 3.5135
Domino Logic Timing	DataStable_M1	M1prech−	0	[0.7215, 2.2620]	OK
	DataStable_A1	A1prech−	0	[0.9520, 1.488]	OK
	DataStable_A2	A2prech−	0	[2.4360, 3.5240]	OK
	DataStable_M2	M2prech−	0	[0.4325, 5.4325]	OK
Controller Protocol Timing	M2done+	A2M−	0	[3.0970, 10.6160]	OK
	A2M+	M2done+	0	[9.9845, 22.3750]	OK
	A1M+	M2done+	0	[10.4500, 16.7400]	OK

Table 2. Timing verification results for typical-case SPICE delays with $\pm 5\%$ variation.

[5] H. Hulgaard. *Timing Analysis and Verification of Timed Asynchronous Circuits*. PhD thesis, University of Washington, Seattle, 1990.

[6] P. Kudva, G. Gopalakrishnan, and E. Brunvand. Performance analysis and optimization for asynchronous circuits. In *Proceedings of the ICCD*, October 1994.

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

[8] K. L. McMillan and D. L. Dill. Algorithms for Interface Timing Verification. In *Proceedings of the ICCD*, October 1992.

[9] C. J. Myers, T. G. Rokicki, and T. H.-Y. Meng. Automatic synthesis and verification of gate-level timed circuits. Technical Report CSL-TR-94-652, Stanford Univ., 1994.

[10] C. D. Nielsen and M. Kishinevski. Performance analysis based on timing simulation. In *Proceedings of the DAC*, June 1994.

[11] C. Tofts. A compositional analysis of the performance of asynchronous pipelines. In *Proceedings of TAU*, November 1995.

[12] E. Verlind, G. de Jong, and B. Lin. Efficient timing analysis of highly concurrent systems. In *Proceedings of TAU*, 1996.

[13] T. E. Williams. Analyzing and improving the latency and throughput performance of self-timed pipelines and rings. In *Proceedings of the ISCAS*, May 1992.

[14] 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 ASYNC*, April 1997.

[15] K. Y. Yun. *Synthesis of asynchronous controllers for heterogeneous systems*. PhD thesis, Stanford University, 1994.

[16] 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 ASYNC*, April 1997.