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

Supratik Chakraborty † Kenneth Y. Yun ‡ David L. Dill †

† 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 circuits designed with implicit timing assumptions. We model interacting asynchronous controllers and datapath elements using timing constraint graphs. Performance metrics and circuit timing constraints to be checked are formulated as time separations between appropriate events. Time separations between all pairs of events are then efficiently computed in a single pass. We present results of analyzing a real asynchronous differential equation solver chip [7] using our proposed technique, thereby demonstrating the practicality of our approach.

1. Introduction

There is mounting evidence that asynchronous circuits are finding niches in high-performance applications, such as Intel's asynchronous instruction length decoder and the asynchronous differential equation solver benchmark circuit [7]. 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 designs and to estimate their performance, efficient and accurate timing analysis techniques are needed. Another interesting problem is to determine how variations in component delays (e.g., due to statistical variations in operating and manufacturing conditions) affect the performance and correctness of these circuits.

Previous work in this area [1, 6, 5, 3, 7] has typically addressed the timing verification and performance analysis problems *separately*. Timing verification of asynchronous circuits has been primarily concerned with controller and inter-controller protocol timing verification. Asynchronous datapaths have traditionally been considered free of timing problems by virtue of their design style. Unfortunately, this is no longer true in modern aggressive asynchronous designs, e.g. [7].

In this paper, we propose a *unified technique* for timing verification and performance analysis of complex asynchronous circuits designed with implicit timing assumptions. We restrict our discussion to choice-free systems, i.e., systems where the next-state is solely determined by the present state and does not depend on external input conditions or choices. Component delays are allowed to vary within specified bounds to model the effect of statistical variations in operating and manufacturing conditions. The temporal behavior of a system of interacting controllers and datapath elements is modeled using timing constraint graphs. Performance metrics and circuit timing constraints are expressed as time separations between appropriate events in the system. Time separations between *all* pairs of events are then computed in *a single pass* using an efficient timing analysis algorithm. We have applied our technique to model and analyze an actual asynchronous differential equation solver chip [7]. This demonstrates the practicality of our approach.

2. A motivating example

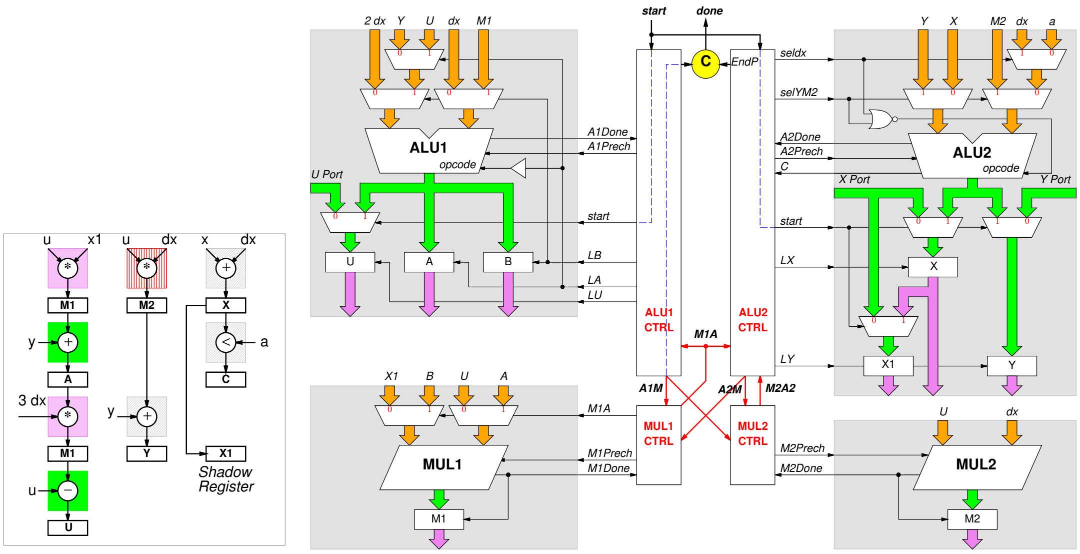
Fig. 1, adapted from [7], shows the core dataflow graph and architecture of a high-performance asynchronous differential equation solver chip (subsequently referred to as Diffeq). The core dataflow graph represent one iteration of computation by the differential equation solver. The solver repeatedly iterates through this dataflow graph until a termination condition is satisfied. We will use this circuit to illustrate our ideas in this paper. The control logic of Diffeq is implemented using four *distributed* asynchronous controllers:ALU1Ctrl, ALU2Ctrl, MUL1Ctrl and MUL2Ctrl. The controllers communicate with one another using a handshaking protocol that implicitly assumes safe timing bounds in signaling. Like synchronous systems, the control is responsible for generating ALU opcodes, MUX selects, register load enables, and precharge/evaluate signals. However, there are implicit assumptions on the timing of datapath signals, e.g., data inputs to domino circuits must stabilize before evaluation begins.

3. Modeling timing behavior

In [7], the DiffEq controllers were designed using the *extended burst-mode (XBM)* asynchronous design style [8]. If we ignore the checks for the termination condition in the state transition diagram of these controllers [7], their behavior is essentially choice-free. This motivates us to analyze systems of choice-free XBM controllers interacting with datapath components. Note, however, that our techniques can be easily extended to model systems with other types of choice-free controllers as well.

We use timing constraint graphs similar to those in [4] to model timing behavior. 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 vertex i is also labeled with a min or max operator, which determines how the time of occurrence of i depends on its immediate predecessors in the graph, e.g., if there exist edges from j

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

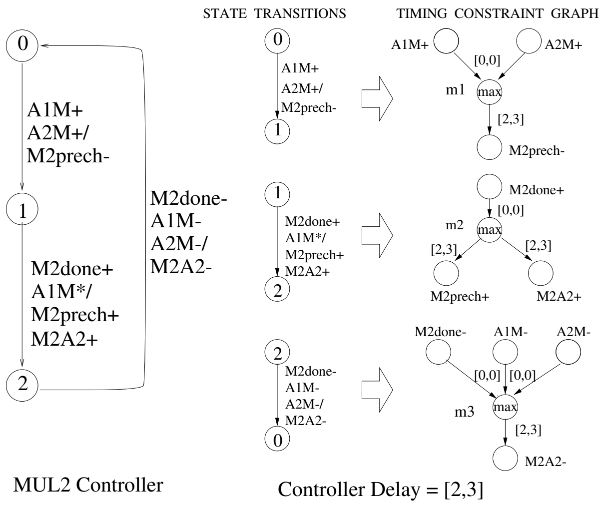


(a) (b)

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

and k to i and if 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.

3.1 Controller timing



(a) (b)

Figure 2. Modeling controller timing.

Fig. 2a shows the state transition diagram of one of the XBM controllers (MUL2Ctrl) in DiffEq. For each state transition, let $\mathcal{I}$ be the set of input transitions and $\mathcal{O}$ the set of output transitions. XBM semantics then require that all *terminating* transitions in $\mathcal{I}$ (input signal transitions ending in $+$ or $-$ in Fig. 2a) must occur before the state transition is enabled, and only then can the transitions in $\mathcal{O}$ fire [8]. To model this, we create a max-type vertex, m, and draw zero-delay edges from the terminating transitions in $\mathcal{I}$ to m (see Fig. 2b). We also draw edges from m to each concurrently enabled transition in $\mathcal{O}$. Each such edge is then labeled with the delay incurred by the controller in generating the corresponding output transition after the last terminating transition in $\mathcal{I}$ has occurred. Note that during the state transition $S_1 \rightarrow S_2$ in Fig. 2a, there is an input transition, A1M$*$, which doesn't end in a $+$ or $-$. This is a special type of transition in XBM circuits called a *directed don't care* [8], and does not affect the state transition $S_1 \rightarrow S_2$. Consequently, the timing constraint graph modeling the $S_1 \rightarrow S_2$ transition does not have A1M$*$ as an input event (see Fig. 2b).

Input events, such as M2done$-$ and A2M$-$ in Fig. 2b, are generated either by datapath components or by other controllers. For example, M2done$-$ is generated by the MUL2 multiplier of Fig. 1b. The signal M2prech$+$ generated by MUL2Ctrl initiates the precharging of the MUL2 multiplier; M2done$-$ then notifies MUL2Ctrl that all internal nodes of the multiplier have been precharged. If the precharging delay is 4 to 5 time units, we can draw an edge from M2prech$+$ to M2done$-$ and label it $[4, 5]$. Unlike M2done$-$, however, A2M$-$ represents a synchronization event generated by ALU2Ctrl and communicated to MUL2Ctrl. This event, therefore, appears as an output event in the timing constraint graph for ALU2Ctrl and also an input event in the graph for MUL2Ctrl. These two events (vertices) may now be connected by a directed edge labeled with the wire delay from ALU2Ctrl to MUL2Ctrl. Continuing this process, timing constraint graphs of the individual controllers can be "stitched" together at the inter-controller synchronization events.

3.2 Datapath timing

In this subsection, we describe the modeling of domino logic arithmetic circuits and edge-triggered registers only due to space constraints. The behavior of other datapath components are modeled similarly.

A domino-logic arithmetic circuit and its timing behavior are depicted in Fig. 3. The Done output is used in asynchronous circuits to indicate completion of precharging Done$-$ and the end of evaluation Done$+$. When the precharge signal is asserted (Precharge$+$), the internal nodes 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 asserts the valid outputs (DataOutValid or Done+) after some data-dependent computation-delay. The events DataInStable and DataInChanged in Fig. 3b represent the stabilization of data inputs prior to the evaluation phase and their subsequent change after the evaluation phase. The behavior of a positive edge-triggered register without asynchronous set and reset inputs is modeled in a similar way in Fig 4. Here, Clock+ represents the rising edge of the clock.

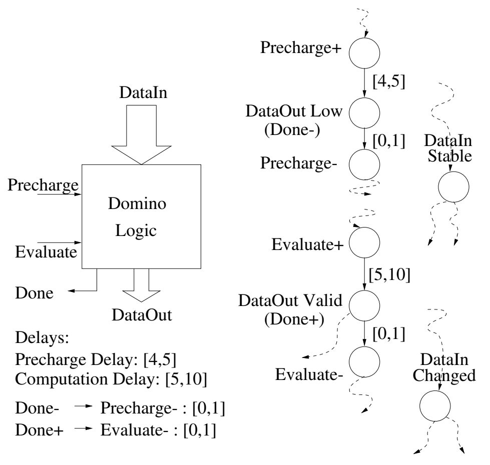


Figure 3. Modeling domino-logic circuit.

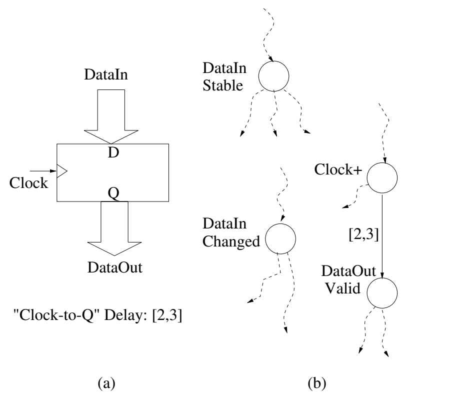


Figure 4. Modeling edge-triggered register.

4 Uniform problem formulation

We now formulate circuit timing constraints and performance metrics 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 simultaneously.

4.1 Performance Analysis

We focus on two important performance metrics of Diffeq: (i) *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 simply determine the time separation between the start of computation in one iteration, and the end of computation in the same iteration. To determine the cycle-time, we compute the time separation between the start of computation in one iteration and the start of computation in the next iteration of the dataflow graph.

4.2 Timing 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 and determine the minimum separation between events DataInStable and Clock+ (for setup-time) and between Clock+ and DataInChanged (for hold-time). In order to verify the timing of domino-logic circuits, we determine the minimum separation between the stabilization of inputs and the arrival of the evaluate signal, and check if the data stabilizes before evaluation begins. Note that if the input data lines always rise monotonically, this requirement is not necessary for correct functioning of the domino circuit. 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 the timing constraints of the inter-controller communication protocol. As an example, consider the state transition diagram of MUL2Ctrl 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 implies that A2M, once it rises, must not fall until M2done rises, i.e., the minimum separation between M2done+ and A2M− must be positive. All other timing constraints in the inter-controller communication protocol of Diffeq can be formulated similarly as time separations between events.

Note that each timing constraint considered in this section involves events which do not have an edge between them in the original timing constraint graph. Therefore, we need to analyze the original graph in order to determine the required time separations.

5 Analysis

We compute bounds on the time separations between events using the algorithm of [2]. The basic idea of this algorithm is as follows. Timing constraints expressed in a timing constraint graph are often non-linear (e.g., min and max). The feasible space of such a system of constraints is non-convex, in general. Hence, efficient 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. Then, it efficiently computes bounds on the time separation of events in this approximate convex space. While the computed bounds are conservative in the worst-case, experiments have shown that they are fairly accurate and often exact in practice [2].

Practical systems of controllers

Most practical systems of choice-free controllers are tightly-coupled in the sense that in steady-state, each controller synchronizes with *all other* controllers, directly or indirectly, between two successive iterations through its state

	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
Controller Protocol Timing	M2done+	A2M-	0	[3.0970, 10.6160]	OK
	A2M+	M2done+	0	[9.9845, 22.3750]	OK

Table 1. Timing verification results for typical-case SPICE delays with $\pm 5\%$ variation.

transition loop. For such systems, we first construct the timing constraint graph of each controller considering *two* iterations of its state transition loop. The cyclic nature of the state transitions insures that the sink events (no outgoing edges) in this graph represent the same signal transitions as the source events (no incoming edges), although they occur two complete iterations later. The graphs for the individual controllers are then linked using synchronization events as explained in Section 3.1, thereby producing the timing constraint graph for the complete system.

We now perform time separation analysis on this graph (for the complete system), pessimistically assuming that the initial separation between every pair of source events is $[-\infty, +\infty]$. Despite this pessimistic approximation, it can be shown that if all component delays are finite, the tight-coupling property and the fact that we unrolled each state transition loop twice, causes the time separations between sink events to be finitely bounded. We now feed these computed bounds back as estimates of the initial separations between the source events and repeat the analysis. If we continue this process repeatedly, it can be shown that the time separation interval between any two sink events monotonically shrinks towards a steady value (i.e., steady upper and lower bounds) with each pass of analysis. For Diffeq, the steady value is obtained in just 2 passes. In general, if k passes of analysis are required to obtain the steady bounds on the time separations, these bounds hold only after each controller has cycled through its state transition loop $2(k-1)$ times. This is because each pass analyzes 2 iterations of each state transition loop. Therefore, for timing verification, we also analyze the initial behavior of the system (from power up until each controller has cycled through its state transition loop $2(k-1)$ times), besides performing the steady-state analysis.

6 Experimental Results

We have modeled and analyzed the complete asynchronous differential equation solver chip 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 variations of component delays affect the circuit's correctness and performance.

We present a subset of our timing verification results in Table 1. Each entry lists a pair of events, the required minimum separation between them and the computed bounds on the separation. All times are in ns. Table 2 shows the latency and cycle time of the differential equation solver obtained with 0%, $\pm 5\%$ and $\pm 10\%$ variations in delays. The 0% entries show the latency and cycle time variation due to data dependency of delays, while the other entries show the added variation in performance due to small variations in component delays (due to manufacturing and operating condition variations). The entire analysis takes less than 0.2 second on an UltraSPARC-1 (143 MHz) with 128 MB of memory. All the computed bounds are exact, as was verified by running an exact algorithm [4] on the same specification. This, however, took 40524.47 seconds on the same platform.

Latency	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 2. Performance analysis results.

7 Conclusion

In this paper, we have presented an efficient technique for timing verification and performance analysis of complex systems with interacting asynchronous circuits. We have also demonstrated the practicality of our approach by analyzing an actual chip. We believe that techniques such as these are indispensable for the design of timing-critical high-performance circuits.

Acknowledgment

We are thankful to J. Arceo of UC, San Diego for helping us with the SPICE simulations.

References

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

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

[3] H. Hulgaard. *Timing Analysis and Verification of Timed Asynchronous Circuits*. PhD thesis, Univ. of Washington, 1995.

[4] K. L. McMillan and D. L. Dill. Algorithms for Interface Timing Verification. In *Proc. of ICCD*, 1992.

[5] 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.

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

[7] 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 *Proc. of ASYNC*, 1997.

[8] K. Y. Yun and D. L. Dill. Unifying synchronous/asynchronous state machine synthesis. In *Proc. of ICCAD*, 1993.