POLYNOMIAL-TIME TECHNIQUES FOR APPROXIMATE TIMING ANALYSIS OF ASYNCHRONOUS SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Supratik Chakraborty
August, 1998

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

David L. Dill
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Giovanni De Micheli

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Kenneth Y. Yun

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Krishna Saraswat

Approved for the University Committee on Graduate Studies:

To my mother

Acknowledgments

I am indebted to several people, without whose help this thesis would never have been written. Foremost among them is my adviser, Professor David Dill. I would like to thank him for his advice, support, patience and encouragement and for giving me the freedom to work on a research topic that interested me. From day one, it has been an enlightening experience doing research under his supervision. He has taught me how to think about difficult research problems, how to sort out the important issues, and how to articulate complicated results clearly. Despite being the only student in his research group working on asynchronous timing analysis, he has always given me all the time, encouragement, technical feedback and support that I needed. It has been my privilege working with him and learning from him.

I am also grateful to my associate adviser, Professor Giovanni De Micheli, for his help and guidance and for being on my oral examination committee as well as on my dissertation reading committee.

I would like to thank Professor Kenneth Yun of the University of California, San Diego for giving me the opportunity to work with him closely for the past three years. He has been a constant source of encouragement for me. The 10 months that I spent at San Diego working in his group were among the most productive of my Ph.D. years. He has been my friend and mentor at the same time.

I am also grateful to Professor Krishna Saraswat for generously agreeing to chair my oral examination and also for reading this thesis.

It has been a pleasure knowing the members of both Professor Dill's and Professor Yun's research groups. At Stanford, Clark Barrett, Jeffrey Su, SeungJoon Park, Jens Skakkebæk, Han Yang, Satyaki Das, Shankar Govindaraju, Uli Stern, Robert Jones, Norris

Ip, Alan Hu, David Park and Vincent Lo were always there to help me in times of need. I am particularly thankful to them for helping me prepare my oral examination. At San Diego, Kevin James, Ayoob Dooply, Viet Do, Julio Arceo and Mike Clovis made my stay so much more enjoyable.

I would like to thank Dr. Pasupathi Subrahmanyam and Professor Ganesh Gopalakrishnan for providing me encouragement and valuable technical advice on several occasions. I am also grateful to Professor Anoop Gupta, with whom I had worked during my first year at Stanford, for believing in my abilities and for helping me find a research topic that interested me.

I would like to express my thanks to Lilian Betters, Helen Nichols, Kersten Barney and Kathleen DiTommaso for taking care of all administrative matters and for helping me out of last-minute snafus in which I find myself so often.

I am grateful to Charles Orgish and Thoi Nguyen for always being there to help me out when I had computer related problems.

I have been extremely fortunate to have had the company of a wonderful set of friends, who provided much needed distractions to my academic life at Stanford. To them, I owe a special acknowledgment. To Salilda and his family, to Dipadi, Suryasish, Subir and Rebecca, I can only say "Thank You" for lack of a better word. To Venu and Nandini, I am thankful for your friendship. I would also like to thank my other friends – Prasenjit, Rini, Bhaskar, Vamsi, Navin, Suhas, Sundar, Tirthankar, Birdy and several others for making my stay at Stanford so much more enjoyable.

Finally and most importantly, I am grateful to my family, especially my mother and aunts, for their love, care, encouragement and support. Without their selfless sacrifices, it would never have been possible for me to pursue my academic goals.

The financial support for this work came from Semiconductor Research Corporation, Contract nos. 95-DJ-389 and 96-DJ-389, and from gifts from Sun Microsystems and Intel Corporation. Their help in this research is gratefully acknowledged.

Supratik Chakraborty
August 1998

Abstract

As designers strive to build systems on chips with ever diminishing device sizes, and as clock speeds of gigahertz and above are being contemplated, the limitations of synchronous circuits are beginning to surface. Consequently, there has been a renewed interest in asynchronous design techniques that use judicious timing assumptions to obtain fast circuits with low hardware overhead. However, the correct operation of these circuits depend on certain timing constraints being satisfied in the actual implementation. Since statistical variations in manufacturing conditions and operating conditions result in uncertainties in component delays in a chip, it is important to analyze asynchronous systems with uncertain component delays to check for timing constraint violations and to determine sufficient conditions for their correct operation. Unfortunately, several timing analysis problems are computationally intractable when component delays are uncertain but bounded. This thesis presents polynomial-time techniques for approximate timing analysis of asynchronous systems with bounded component delays. Although the algorithms are conservative in the worst case, experiments indicate that they are fairly accurate in practice.

Three important problems in asynchronous timing analysis are addressed. First, a polynomial-time algorithm for computing bounds on signal propagation delays from each primary input to each gate in a combinational circuit is described. This has applications in determining input timing constraints for correct operation of asynchronous circuits. To improve the accuracy of simulation, a polynomial-time reconvergent fanout analysis technique is also proposed. As an application, a timing analysis tool for a class of asynchronous circuits is presented. Experiments indicate that the proposed algorithm is efficient and fairly accurate in practice.

Next, the problem of computing bounds on time separation of events in asynchronous systems is addressed. This has important applications in timing verification, analysis and optimization of asynchronous circuits. A polynomial-time algorithm for analyzing choice-free systems with min and max timing constraints, but without repeated occurrences of events, is proposed. The algorithm is based on computing a convex over-approximation of the feasible region of a system of min and max constraints. The algorithm is exact when there is a single source event in the system and all timing constraints are either max-only or min-only. For systems with both min and max constraints, the computed bounds are conservative in the worst case. Nevertheless, experiments indicate that the algorithm is fairly accurate in practice, even when both min and max constraints are present.

Finally, the time separations problem for choice-free systems with repeated events and min and max timing constraints is addressed. Tightly-coupled systems – a class of practical asynchronous systems – are characterized, and a polynomial-time algorithm for computing time separation bounds in tightly-coupled systems is proposed. It is shown that finite bounds on the long-term time separation of events in tightly-coupled systems can be obtained even when the initial startup behavior of the system is unknown. The proposed algorithm is iterative, and an upper bound on the number of iterations required for the analysis to converge to a fix-point is provided for systems in which all component delays are bounded by integers. To demonstrate the practicality of the approach, a complete asynchronous differential equation solver chip has been analyzed using the proposed algorithm.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

There are two broad paradigms of digital design – *synchronous* and *asynchronous*. A synchronous circuit is one in which all operations are sequenced by one or more globally distributed periodic signals called *clocks*. An asynchronous circuit, on the other hand, does not have a global clock to synchronize its actions; instead, it uses signaling protocols between components to achieve synchronization and ensure proper sequencing of operations. Synchronous circuits are easier to design, analyze and validate than their asynchronous counterparts; consequently, they have dominated digital design for the past three decades. However, as designers strive to build systems on chips with ever diminishing device sizes, and as clock speeds of gigahertz and above are being contemplated, the limitations of synchronous circuits are beginning to surface. In this setting, asynchronous circuits hold some promise, especially in eliminating clock skew problems, lowering power consumption, providing average-case performance instead of worst-case performance, providing greater robustness to component delay variations, etc. Consequently, there has been a revival of interest in asynchronous design techniques over the last decade and a half.

Traditional asynchronous design styles make conservative assumptions about the delays of circuit components, and generate circuits that function correctly regardless of the actual component delays. However, the robustness is usually achieved at the cost of significant hardware overhead and at times, degraded performance. This renders these circuits unsuitable for use in most practical applications. Practical asynchronous circuit designs, therefore, make judicious timing assumptions and exploit available knowledge of signal transition timing in order to enable important design optimizations. This results in faster circuits with lower hardware overhead. However, the correct operation of these circuits depend on certain timing constraints being satisfied in the actual implementation. The problem is exacerbated by the fact that statistical variations in manufacturing and operating conditions result in uncertainties in component delays in a chip. To check whether all timing constraints are satisfied for all possible component delay variations and to determine conditions for correct operation of the circuits, efficient timing analysis techniques for asynchronous circuits are needed.

1.1 Delay Models

The behavior of an asynchronous circuit critically depends on the delays of its components. Below, an overview of the delay models commonly used for asynchronous circuits is provided, with emphasis on the model used in this thesis.

- *Fixed delay*: In this model, the delay of every component is assumed to be fixed. Since it is almost impossible to obtain the precise delay of a component in a chip, this is not a realistic model for timing verification purposes. However, it has been used by some researchers, such as Burns [18] and Lee [63], for computing performance metrics of asynchronous systems by treating the average delay of each component as its fixed delay.

- *Unbounded delay*: In this model, each component is assumed to have a delay that can assume any finite non-negative value. While design procedures based on this delay model produce circuits that are extremely robust to delay variations, this model is unrealistically conservative.

- *Bounded delay*: In this model, every component is assumed to have an uncertain delay that lies between given upper and lower bounds. The delay bounds take into account potential delay variations due to statistical fluctuations in the fabrication process, variations in ambient temperature, power supply, etc. This is the most realistic of the three delay models, and is the one used in this thesis.

> The problem of characterizing delay bounds is an interesting problem in itself. The bounds may be obtained by simulating each component under best case and worst case processing and operating conditions. Alternatively, they may also be obtained by adding conservative margins to the nominal delays provided by the chip manufacturer. In any case, this thesis does not address the problem of characterizing delay bounds; instead, it is assumed that the delay bounds are given.

Another important aspect of the delay model is *correlation* of component delays. Since a change in processing or operating conditions affects several (if not all) components in a chip, component delays in a chip tend to be correlated. However, for simplicity of analysis, this thesis assumes that component delays are independent of each other. Note that this can cause the results of timing analysis to be more conservative than they are in reality.

Delays may also be classified as *pure* or *inertial*. A pure delay simply shifts a waveform in time without altering its shape. In contrast, an *inertial* delay may alter the shape of a waveform by suppressing narrow pulses. For simplicity, all delays in this thesis are assumed to be pure delays.

1.2 Classes of Asynchronous Circuits

Asynchronous circuits are conventionally classified based on the delay model used by them. An overview of the main approaches is provided below. For a more complete discussion, the reader is referred to the literature [43].

Delay-insensitive (DI) circuits assume unbounded delays for both gates and wires; hence, they are very robust to component delay variations. The literature contains a large body of work on DI circuits [26, 77, 87, 35, 84, 13]. However, it has been shown that if one uses gate libraries with only single-output gates, the class of DI circuits is severely restricted to those that use only the *Muller C-element*, buffers and inverters [68]. Practical DI circuits are, therefore, built using more complex multiple-output components [77, 34]. Although these components can be composed in a delay-insensitive manner, timing constraints must be satisfied internal to each component for it to function correctly.

A *quasi-delay-insensitive (QDI)* circuit assumes that both gate and wire delays are unbounded, but adds the restriction that if a wire forks out to multiple destinations, then the delays along all branches of the fork are identical. This is traditionally called the *isochronic fork* assumption [66]. A *speed-independent (SI)* circuit is one which assumes that gate delays are unbounded but wires have negligible delay compared to gates. SI circuits were pioneered by Muller (see [75]) and subsequently, there has been significant work both on the theoretical and practical aspects of SI and QDI circuits [32, 6, 55, 25, 74, 90, 67]. The advantage of QDI and SI circuits over DI circuits is that they can be technology-mapped to libraries of basic gates like AND, OR, NAND, NOR, etc. However, the circuits must be carefully laid out so that the isochronic fork assumption or zero wire delay assumption is met.

A large number of practical asynchronous circuits fall outside the categories of DI, SI or QDI circuits. These circuits make use of bounded delay assumptions to reduce the hardware overhead and enable faster operation. An important class of such circuits is *fundamental-mode* circuits. A fundamental-mode circuit assumes that the environment waits long enough for the circuit to stabilize before responding to a change in the circuit's outputs. Bounded delays of gates and wires in the circuit are used to determine an upper bound on the period the environment must wait for the circuit to stabilize. The literature contains a large volume of work on fundamental-mode circuits, starting from Huffman, Unger and McCluskey [44, 45, 89, 70, 27, 83, 82, 99].

Timed circuits [79] are another class of practical asynchronous circuits that make use of bounded delay assumptions within the circuit and in the environment. This can lead to significant improvement in circuit complexity and performance compared to DI, SI or QDI circuits [78]. However, as with fundamental-mode circuits, the correct operation of these circuits depend on the validity of the delay bounds in the actual implementation. The reader is referred to the literature [61, 8, 43] for a more thorough discussion of other practical asynchronous design styles that make use of bounded delay assumptions.

1.3 Why Approximate Timing Analysis?

We have seen above that practical asynchronous circuits depend on certain timing constraints for their correct operation. Since component delays can assume any values within their bounds, we need timing analysis techniques that can reason about circuits with bounded delays and check if all timing constraints are satisfied for all delay variations. While the accuracy of the analysis is important, it is equally important that the analysis be efficient for it be useful in practice. Without efficient timing analysis algorithms, it is hard to contemplate the acceptance of asynchronous systems in large, real-world applications. Therefore, a balance between efficiency and accuracy must be struck. In particular, polynomial-time timing analysis techniques that are fairly accurate in practice, though approximate in the worst case, is an interesting alternative to explore.

A large number of bounded-delay timing analysis techniques for asynchronous circuits are reported in the literature. Detailed references to earlier work are given in the "Related Work" sections of Chapters 2, 4 and 5. Techniques that analyze asynchronous circuits at the gate level usually compute bounds on signal propagation delays through the circuit. On the other hand, techniques that analyze asynchronous systems at higher levels of abstraction usually involve determining bounds on the time separations between events. Unfortunately, both these problems are known to be computationally intractable in general. Ishiura [50] has shown in the context of hazard detection that finding exact bounds on signal propagation delays through a circuit is computationally intractable. Similarly, McMillan and Dill [72] have shown in the context of interface timing verification that finding exact bounds on the time separation of events is computationally intractable in general. Therefore, exact timing analyzers, based on either timing simulation or time separations analysis, have worst-case running time that are at least exponential in the size of the system (unless $P = NP$).

Unfortunately, algorithms with worst case exponential complexity are not always well-suited for practical applications. This is particularly true when large systems are analyzed repeatedly, as in a design-analyze-redesign environment. An approximate timing analysis technique that has a low-order polynomial time complexity and computes fairly accurate results in practice may be more useful in such situations. Polynomial-time approximate algorithms can also be used as a fast filter to quickly narrow down the set of potential timing problems in a circuit. Once a small set of potential problems are identified, more expensive but exact timing analysis algorithms can be used to investigate each problem in greater detail. This can lead to a pragmatic approach to timing analysis of large and complex systems. It must be noted that since polynomial-time algorithms for computationally intractable problems must necessarily be approximate in the worst case, they may report conservative results, such as timing violations that never actually occur. However, by carefully designing the algorithms, such occurrences can be minimized in real applications. This motivates the investigation of polynomial-time timing analysis techniques that are approximate in the worst case, but fairly accurate in practice.

1.4 Contributions

This thesis shows that polynomial-time algorithms for approximate timing analysis of asynchronous systems can be fairly accurate in practice. Consequently, they hold a lot of promise for practical timing analysis of large and complex asynchronous designs.

Specifically, this thesis presents polynomial-time approximate algorithms for three important timing analysis problems, namely *min-max timing simulation*, *time separation of events in acyclic systems* and *time separation of events in cyclic systems*. To demonstrate the practicality of the algorithms, they have been applied to several asynchronous systems, including a suite of *3D* asynchronous benchmarks [99], and a complete asynchronous differential equation solver chip [101].

- *Min-max timing simulation*: A polynomial-time algorithm has been proposed for computing bounds on signal propagation delays from each primary input to each gate in a combinational circuit. To improve the accuracy of simulation, a polynomial-time *reconvergent fanout analysis* technique is also described. The proposed algorithm addresses the shortcomings of previous polynomial-time algorithms (and some exponential time algorithms) for min-max timing simulation, and produces fairly accurate results in practice.

- *Timing analysis tool for 3D circuits*: The min-max timing simulation algorithm mentioned above has been used to design an automatic timing analysis tool for a class of asynchronous circuits, called *extended burst-mode circuits*, implemented in the 3D design style [99]. The tool accepts as inputs a gate-level 3D circuit annotated with gate delay bounds, and the corresponding extended burst-mode state transition diagram, and extracts safe bounds on timing constraints required for correct operation of the circuit.

- *Time separation of events*: Finding bounds on the time separation of events is a fundamental problem in the analysis, optimization and verification of asynchronous systems. However, the problem is computationally intractable when both min and max type timing constraints are present. Existing algorithms either exclude min constraints for efficiency of analysis, or blow up exponentially in the worst case when both min and max constraints are present. This thesis presents polynomial-time algorithms for computing approximate bounds on the time separation of events in systems with both min and max constraints. First, the problem is addressed for systems without repeated occurrences of events, called acyclic systems. Then, an algorithm for systems with repeated occurrences of events, or cyclic systems, is proposed. Although the computed bounds are approximate in the worst case, experiments indicate they are fairly accurate in practice. In addition, the running times of the proposed algorithms compare very favorably with those of an existing exact algorithm.

  To demonstrate the practicality of the proposed approach, a complete asynchronous differential equation solver chip [101] has been modeled and analyzed using the proposed algorithms.

1.5 Thesis Overview

The remainder of this thesis is organized as follows.

Chapter 2 describes a polynomial-time algorithm for approximate min-max timing simulation of combinational circuits, for use in timing analysis of fundamental-mode asynchronous circuits. A reconvergent fanout analysis technique for improving the accuracy of simulation is discussed. Proofs of correctness of the algorithms and experimental results on a suite of asynchronous benchmarks are presented.

Chapter 3 describes the design and operation of a timing analysis tool for extended burst-mode circuits implemented in the 3D design style. The results of analyzing a suite of 3D asynchronous benchmarks are presented.

In Chapter 4, acyclic timing constraint graphs are introduced as a means of representing timing constraints between non-repeated events. A strategy for over-approximating min and max constraints by systems of linear inequalities is proposed. These approximations are then used to design a polynomial-time algorithm for computing conservative bounds on the time separations of all pairs of events. Proofs of correctness of the algorithm along with experimental results confirming the accuracy and efficiency of the algorithm are presented.

Chapter 5 addresses the problem of computing time separation of events in systems with repeated events. Cyclic timing constraint graphs are introduced as a means of representing timing constraints in such systems. An iterative algorithm is proposed for computing conservative bounds on the long-term time separation of events. It is shown that the computed bounds monotonically converge to their fix-point values with every iteration. For systems with non-negative integral component delays, an upper bound on the number of iterations required to converge to the fix-point is given. A proof of correctness of the algorithm is provided. Finally, a complete asynchronous differential equation solver chip is modeled and analyzed using the proposed algorithm.

Finally, Chapter 6 gives some concluding remarks and outlines directions for future research.

Chapter 2

Min-Max Timing Simulation

2.1 Introduction

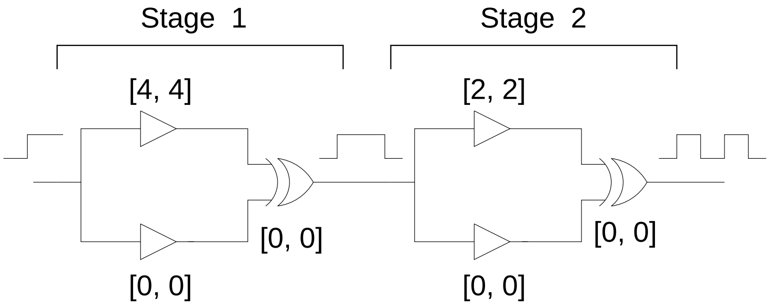
Timing simulation is an important technique for analyzing high-speed digital circuits. Conventionally, each circuit component is assumed to have a fixed delay; given the primary input transition times, a timing simulator computes signal propagation delays from the primary inputs to the primary outputs and other internal points in the circuit. In practice, however, statistical variations in IC fabrication conditions, operating conditions, etc. give rise to uncertainties in component delays in a chip. Timing simulation that determines upper and lower bounds on signal propagation delays, given unknown but bounded component delays, is called *min-max timing simulation*. This chapter describes a polynomial-time algorithm for approximate min-max timing simulation of combinational circuits, for use in timing analysis of *fundamental-mode* asynchronous circuits [89].

Timing analysis techniques for synchronous circuits are not directly applicable to asynchronous circuits because synchronous analyzers work under the assumption that the relative transition times of all circuit inputs are known. For example, it is commonly assumed that all inputs to a synchronous circuit transition at the same time, and successive sets of input transitions appear after fixed time intervals, synchronized with the clock. In contrast, the relative transition times of the inputs to an asynchronous circuit are often not known *a priori*. In fact, it is often necessary to determine constraints on input transition times that ensure correct operation of the circuit. Therefore, we need separate timing analysis techniques that compute bounds on signal propagation delays from each circuit input to each gate, without requiring the user to specify the relative input transition times. These delay bounds can then be used to determine sufficient timing constraints for correct operation of the circuit.

Unfortunately, exact min-max timing simulation is computationally difficult. Ishiura has shown that hazard detection in combinational circuits with uncertain delays is NP-hard when each gate delay is expressed using a fixed number of bits, and PSPACE-complete when the number of bits representing each delay is a polynomial in the size of the circuit [50]. Informally, a *combinational hazard* is an unintended signal transition, such as a $0 \rightarrow 1 \rightarrow 0$ transition when the signal should have remained stable at 0[1]. Since the existence of hazards at the output of a circuit can be detected by exact min-max timing simulation, it follows that exact min-max timing simulation is computationally intractable. The complexity of the problem stems from two sources. First, given a circuit with bounded delays, the number of distinct component delay combinations can be exponential in the number of components. Since the behavior of the circuit depends on the choice of component delays in general, determining exact minimum and maximum signal propagation delay bounds requires examining all delay combinations in the worst case. Second, exponentially long sequences of transitions can result from a single input change in certain circuits. For example, Fig. 2.1 adapted from [30], shows a circuit composed of two cascaded stages. For an n-stage circuit built of such stages, if the delay of the top buffer in stage i is chosen to be 2^{n-i+1} and the delays of all other gates are set to 0, the number of transitions at the output of the i^{th} stage can be shown to be equal to 2^i. There exist other combinations of component delays and other circuits which also exhibit this phenomenon, called *event-proliferation*. Event-driven simulators that keep track of all events generated in the circuit suffer from worst case exponential behavior because of event proliferation.

Fortunately, listing *all* signal transitions is not necessary for most practical purposes. Usually, it is sufficient to determine whether a signal is changing or stable during a temporal window. For example, hazard analysis in asynchronous circuits depends on whether a gate

[1] Formal definitions and classification of hazards may be found in classical textbooks [89, 71].



Gate delays are [min, max]. Wires have zero delay.

Figure 2.1: Example of event-proliferation.

input has spurious transitions on it during the interval when the gate is sensitized to this input. Therefore, a tradeoff between efficiency and accuracy can be made by abstracting sequences of transitions using multi-valued logic and by approximating the earliest and latest signal propagation times. The approximations should be *conservative*, meaning that the analysis should never fail to report an actual timing violation. However, in order to minimize *false alarms* – timing violations that do not actually occur, but are flagged by the analysis – it is also important to improve the accuracy of analysis, while remaining efficient.

This chapter describes a polynomial-time algorithm for computing approximate signal propagation delay bounds from each primary input to each gate in a combinational circuit. Unlike conventional timing simulators, the proposed algorithm does not require the user to specify the relative input transition times; instead, given a set of primary input transitions, the computed bounds apply regardless of the order in which the inputs actually transition. As noted above, this has applications in asynchronous timing analysis, where the relative transition times of the inputs are often not known in advance. To circumvent the event proliferation problem, an abstract 13-valued waveform algebra is proposed. This allows signal transitions to be represented succinctly, without losing information about hazards and initial and final states of signals. The signal propagation delay bounds computed by the algorithm are conservative in the worst case, meaning maximum bounds may be over-estimated and minimum bounds under-estimated. To improve the accuracy, a polynomial-time reconvergent fanout analysis technique is proposed. This helps in detecting ordering between seemingly overlapping signal transitions, which in turn helps in increasing the accuracy of simulation. Experimental results indicate that the accuracy can be significantly improved in certain cases by means of the proposed reconvergent fanout analysis technique.

The remainder of this chapter is organized as follows. Section 2.2 reviews related work on min-max timing simulation. Section 2.3 describes a 13-valued abstract waveform algebra. In Section 2.4, a polynomial-time algorithm for approximate min-max timing simulation of combinational circuits is described. Section 2.5 describes two pathological examples where our algorithm fails to compute exact results. Section 2.6 presents results of applying our algorithm to a suite of asynchronous benchmarks. Finally, Section 2.7 summarizes the contributions of this chapter.

2.2 Related Work

A large number of min-max timing simulators are described in the literature [12, 73, 10, 52, 51, 33, 88, 65, 64, 31]. However, most of these are not well-suited for our purpose because of of the following reasons:

1. They require the user to specify the relative input transition times, which is usually not known in advance in asynchronous circuits, or
2. They have worst case exponential complexity, which makes them unattractive for analyzing large circuits, or
3. They produce overly pessimistic results in the presence of reconvergent fanouts.

This section briefly reviews some representative works.

One of the earliest polynomial-time techniques for approximate min-max timing simulation of combinational circuits is due to Breuer and Friedman [12]. Their technique uses a 5-valued algebra and produces extremely pessimistic results in the presence of reconvergent fanouts. The authors acknowledge this problem and suggest *ad hoc* methods to improve the accuracy of simulation.

The work of Ulrich, Lentz, Demba and Razdan [88] abstracts multiple transitions as *glitches* and single transitions as *edges* and computes separate signal propagation delays from each primary input and internal fanout point to each gate in the circuit. Their algorithm has polynomial-time complexity; however, their simplistic method of eliminating *false glitches* produces overly pessimistic results in the presence of nested reconvergent fanouts.

Time-symbolic simulation (TSS) for hazard detection and timing verification of asynchronous circuits was proposed by Ishiura, Takahashi and Yajima [52]. Their method explicitly considers all combinations of gate delays and enumerates all signal transitions corresponding to each delay combination. Although this produces exact results, the exponential complexity of this technique restricts its applicability to small circuits (up to 100 gates, as reported by the authors [52]).

The idea of *coded time symbolic simulation (CTSS)* was proposed by Ishiura, Deguchi and Yajima [51]. In CTSS, time is discretized and the delays of gates are encoded using binary variables. The output of the circuit is then expressed as a Boolean function of these variables and the primary inputs. The Boolean functions are represented and manipulated using *shared binary decision diagrams (SBDD)* [14, 76]. This provides exact results, but has a worst case complexity that is exponential in the size of the circuit. The running time also depends on the granularity of discrete time, and on the ordering of variables in the SBDDs. Experimental results [51] indicate that the running time increases rapidly with the number of gates, and is significant even for small circuits (350.4s on a Sun3/60 for a 96 gate circuit [51]).

Devadas, Keutzer, Malik and Wang have addressed the problem of verifying gate-level implementations of asynchronous circuits with bounded component delays [31]. Given a circuit implementation, they simulate the circuit to extract the set of behaviors, and then compare the extracted behaviors to the specified behavior. Their simulation algorithm is similar to that of Ishiura et al [52]. The algorithm is exact and has a worst case complexity that is exponential in the circuit size. Pruning techniques [30] can be used to reduce the running time to a certain extent. However, the complexity of simulation is still high and can be exponential in the circuit size even after pruning. Moreover, the effectiveness of pruning depends strongly on the interval during which the circuit outputs are observed: widening the interval of observation can result in a significant increase in the running time. Since transitions occurring at all time are potentially important when analyzing asynchronous circuits, the effectiveness of pruning in asynchronous timing analysis techniques seems to be limited. It must be noted, however, that pruning was originally developed [30] as a means of speeding up timing simulation of synchronous circuits, so it is not surprising that its effectiveness in asynchronous timing analysis techniques is limited.

In his Ph.D. thesis, Lindermann [64] describes a sophisticated timing simulator, called MTV, and compares it to an earlier simulator, called SCALD [73]. MTV uses an event-graph to remember causal relations between events and detect orderings between them. Unfortunately, the number of events in the event-graph can be exponential in the size of the circuit in the worst case; consequently, MTV has worst case exponential complexity. Moreover, MTV does not guarantee exact bounds on signal propagation delays in all cases, despite its worst case exponential complexity.

Martello describes a min-max timing analyzer, called HDTV in [65]. It has been shown [65] that the worst case complexity of HDTV is roughly 2^{n^n} where n is the number of wires in the circuit. This makes it computationally prohibitive to use it even on small circuits.

2.3 13-Valued Waveform Algebra

This section describes a 13-valued abstract waveform algebra to represent signal transitions succinctly. This allows us to circumvent the event-proliferation problem by intentionally "forgetting" detailed information about multiple transitions on a signal. Although the details are somewhat different from previous work, this work is essentially a continuation of a long tradition of hazard analysis using multi-valued logic. It was Eichelberger who first used ternary logic simulation for detecting hazards in sequential circuits [36]. Subsequently, multi-valued logic has been used in different contexts [11, 38, 12, 58, 23]. The 13-valued algebra proposed in this section most closely resembles that used by Chakraborty, Bushnell and Agrawal [23] for delay fault test generation.

The following discussion assumes that we are given a combinational circuit with gates of known functionality. The waveform on each wire or gate output is represented using a triple $\langle b, m, e\rangle$. The components b and e represent the initial and final states of the signal and are assigned values from 0, 1 and X (X represents 0, 1 or changing repeatedly). The m component represents the intermediate behavior of the signal and is assigned a value from 1 (stable high), 0 (stable low), $\uparrow$, $\downarrow$ (single rising/falling transition) and X (potentially multiple transitions). Using this representation, there are two *constant* values: $\langle 1, 1, 1\rangle$ and $\langle 0, 0, 0\rangle$; two *clean* transitions: $\langle 1, \downarrow, 0\rangle$ and $\langle 0, \uparrow, 1\rangle$; four *hazards* $\langle 0, X, 0\rangle$, $\langle 1, X, 1\rangle$, $\langle 0, X, 1\rangle$ and $\langle 1, X, 0\rangle$; one *undefined* signal, $\langle X, X, X\rangle$; two signals that are undefined to begin with but eventually settle to binary values, $\langle X, X, 1\rangle$ and $\langle X, X, 0\rangle$; and two signals that have binary values to begin with but eventually become undefined, $\langle 1, X, X\rangle$ and $\langle 0, X, X\rangle$.

Every Boolean function can be extended to operate on values from the 13-valued algebra. Consider an assignment of values $\langle b_i, m_i, e_i\rangle$ to n input variables of a Boolean function f. A *trajectory* is defined to be a sequence of assignments of Boolean values to the input variables, where exactly one variable changes value from one element of the sequence to the next. The assignment of 13-valued waveforms to the inputs represents a set of trajectories in the points of an n-dimensional Boolean hypercube. Each trajectory starts in a subcube defined by the vector of start values (b_i for each input i) and ends in a subcube defined by the vector of end values (the e_i values).

When an input variable has a constant binary value for m_i, the Boolean value of the variable must have the value m_i at every point in the trajectory. If $m_i = \uparrow$ or $\downarrow$, the Boolean variable must change exactly once in the proper direction along every trajectory. If $m_i = X$, the Boolean value of the variable can change arbitrarily in the trajectory. Let $\langle b_f, m_f, e_f\rangle$ represent the value of the Boolean function f on the 13-valued input values. If f is constant throughout the start subcube, b_f is the constant value. Otherwise, b_f has the value X. The definition of e_f is similar. If the value of f throughout every trajectory is a constant 1 or 0, m_f has that value. If f changes from 0 to 1 (1 to 0) exactly once on every trajectory, m_f has the value $\uparrow$ ($\downarrow$). Otherwise, m_f has the value X.

As an example, consider a 2-input AND function, with input waveforms $\langle 0, \uparrow, 1\rangle$ and $\langle 1, \downarrow, 0\rangle$. Fig. 2.2 shows the 2-dimensional Boolean hypercube with the start and end subcubes circled (the binary values of the inputs are shown as an ordered pair). The AND function evaluates to 0 in both these subcubes; so, $b_f = e_f = 0$. The trajectories corresponding

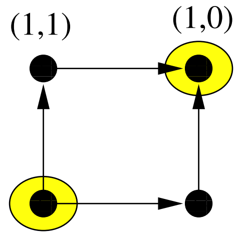


Figure 2.2: Illustrating trajectories in 13-valued simulation.

to the input waveforms are: $(0,1) \rightarrow (0,0) \rightarrow (1,0)$ and $(0,1) \rightarrow (1,1) \rightarrow (1,0)$, as shown in Fig. 2.2. The AND function evaluates to 0 at all points in the first trajectory. However, it changes from 0 to 1 and then to 0 along the second trajectory. Therefore, the value X is assigned to m_f, and the 13-valued output of the AND function is $\langle 0, X, 0 \rangle$. For efficiency, the behavior of standard gates on all 13-valued inputs can be precomputed and stored in a table. 13-valued simulation of a combinational circuit then proceeds in the obvious way: gates are processed in topological order and the 13-valued waveform at the output of each gate is determined from the waveforms on its inputs by looking up the table.

2.4 A Polynomial-Time Algorithm

This section describes a polynomial-time algorithm for computing approximate bounds on signal propagation delays from each primary input to each gate in a combinational circuit. The gate delay model is first described, followed by the basic min-max timing simulation algorithm. A reconvergent fanout analysis technique for improving the accuracy of the basic algorithm is also proposed.

2.4.1 Gate Delay Model

The gate delay model is as shown in Fig. 2.3. For each gate, each input waveform is delayed by the appropriate input-to-output delay, and a zero-delay functional block is then used to generate the output waveform. Wire delays are modeled by inserting a delay buffer along each wire. For simplicity of notation, it is assumed that the delay from an input p of gate G to its output lies in the interval $[d_{p,G}, D_{p,G}]$. It should be understood that this is actually a function of the waveforms (rising/falling), output capacitance, etc.

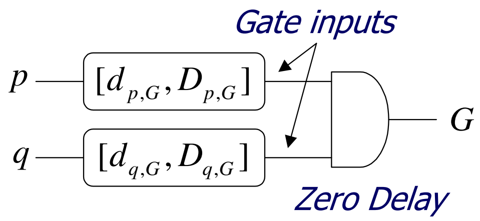


Figure 2.3: Gate delay model.

As noted in Chapter 1, all delays considered in this discussion are pure delays, and the delays of gates are assumed to be uncorrelated. Although some correlation can potentially be modeled by the *tracking delay model* of Lam and Brayton [59], this is not considered in this discussion.

2.4.2 Basic Algorithm

This subsection describes the basic min-max timing simulation algorithm. Combinational circuits composed of basic gates (AND, NAND, OR, NOR, NOT and delay buffers) are considered. For notational convenience, the output of a gate G is referred to by the same label, G, and each primary input is viewed as a gate with zero inputs. For each gate, the inputs of the zero-delay functional block in Fig. 2.3 are referred to as the *gate inputs*. Thus, if the output of gate G_1 is connected to an input of gate G_2, we distinguish between the output of G_1 and the gate input of G_2. This is illustrated in Fig. 2.4.

There are two inputs to the timing simulator: (i) a combinational circuit with delay bounds annotated on each gate, and (ii) an input stimulus consisting of 13-valued waveforms associated with the primary inputs. The output of the simulator is an assignment of 13-valued waveforms to all gates in the circuit. In addition, each gate is annotated with a set of intervals, one for *each primary input* i, giving bounds on the signal propagation delay from i to the output of the gate *for the given input stimulus*.

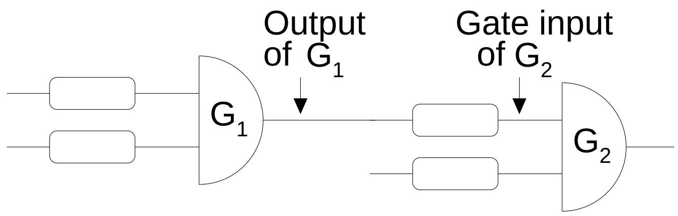


Figure 2.4: Illustrating gate output and gate input.

The following data structures are associated with each gate G. They are updated as the algorithm executes:

- A $value$ field which stores a 13-valued waveform.
- An array $del[1 \ldots max\_primary\_inputs]$ of tuples (t, T). $del[i].t$ stores a lower bound on the signal propagation delay from primary input i to the output of G, and $del[i].T$ stores an upper bound on the same delay. If a transition on primary input i does not cause a transition on the output of G, the tuple $(+\infty, -\infty)$ is assigned to $G.del[i]$.

The top-level algorithm is shown in Fig. 2.5. For each gate, a 13-valued output is first determined by looking up a precomputed table. If the result is a constant value, 13-valued evaluation guarantees that for all possible orderings of the gate input transitions, the constant value is produced at the output of the gate; so no further timing analysis is required. Otherwise, a more detailed and costly analysis is performed by function MinMaxAnalyze.

To understand how function MinMaxAnalyze works, the potential sensitization of a gate is defined as follows. Gate G is *potentially sensitized* to a transition on gate input p if the transition on p potentially causes a transition on the output of G. Note, however, that p need not necessarily cause G to transition. The action of function MinMaxAnalyze can then be summarized as follows. For each gate input p of G, it is determined whether G is potentially sensitized to the transition on p. If it is, the signal propagation delays from the primary inputs to p are used to update the corresponding delays from the primary inputs to G.

$\underline{\mathsf{Top-levelalgorithm}}$

1. Sort gates topologically;
2. (a) **for each** gate G (includes primary inputs)
   for each primary input i
   $G.del[i] = (+\infty, -\infty)$; */* G not affected by transition on i */*
   (b) **for each** transitioning primary input i
   $i.del[i] = [0, 0]$;
3. **for each** gate G in topological order
   (a) $\mathsf{LookUpThirteenValuedTable}(G)$;
   (b) **if** $(G.value \neq \langle 000 \rangle$ or $\langle 111 \rangle)$
   $\mathsf{MinMaxAnalyze}(G)$;

$\underline{\mathsf{LookUpThirteenValuedTable}}(G:$ gate$)$
$G.value =$ 13-valued output from lookup table indexed by
13-valued inputs of G.

Figure 2.5: Top-level algorithm.

MinMaxAnalyze(G : **gate**)

for each transitioning gate input p of G

1. **for each** other gate input q of G
   (a) Determine if transitions on p and q are ordered;
   (b) Using ordering information:
   for each primary input i that affects p
   (i) $qep_i =$ most-sensitizing-value of q at the earliest time a transition on i reaches p;
   (ii) $qlp_i =$ most-sensitizing-value of q at the latest time a transition on i reaches p;
2. **if** (G has a potential hazard)
   Use ordering information between gate input transitions to determine if hazard is masked.
3. **if** (hazard is masked)
   (a) Update G.value to hazard-free waveform.
   (b) **if** (new G.value == $\langle 000 \rangle$ or $\langle 111 \rangle$)
   $G.del[i] = (+\infty, -\infty)$ for all primary inputs i;
4. **else**
   for each primary input i that affects p
   /* Let gate input p be connected to the output of gate G' */
   (a) Using qep_i values:
   if G is potentially sensitized to transition on p
   $p.del[i].t = G'.del[i].t + d_{p,G}$;
   $G.del[i].t = \min(G.del[i].t,\ p.del[i].t)$;
   (b) Using qlp_i values:
   if G is potentially sensitized to transition on p
   $p.del[i].T = G'.del[i].T + D_{p,G}$;
   $G.del[i].T = \max(G.del[i].T,\ p.del[i].T)$;

Figure 2.6: Basic min-max timing analysis algorithm.

To determine whether G is potentially sensitized to p, we need to know all possible combinations of values on the other gate inputs *at the time when p transitions*. In general, this requires examining all possible orderings between the transitions on all gate inputs. Unfortunately, this is not efficiently solvable for circuits with reconvergent fanouts – the number of transitions on each gate input can be exponential in the size of the circuit, and the number of orderings between transitions can grow combinatorially with the number of transitions in the worst case. Therefore, for efficiency, a conservative approximation is used. Essentially, every gate input q that is different from p is considered one at a time, and the algorithm tries to determine if q starts transitioning after p has stabilized to its final value or vice versa. If such an ordering cannot be detected, it is conservatively assumed that the transitions on p and q could overlap. However, if an ordering is detected, the ordering information is used to determine if G is potentially de-sensitized to p or if hazards on the output of G are masked.

The simplest strategy for detecting an ordering between p and q is to check if both p and q are affected by transitions propagating from only one primary input i. If they are, signal propagation delay bounds from i to p and q can be used to detect ordering of transitions. However, this can produce overly pessimistic results since information about reconvergent fanouts is ignored. For example, if both p and q are on the fanout branches of the same signal, the transition times of p and q are correlated. However, the simple strategy outlined above ignores this correlation. In addition, if multiple primary inputs affect p or q or both, the above strategy cannot be used to predict an ordering of transitions on p and q because the relative transition times of the primary inputs are unknown. Therefore, the naive strategy may not be very useful in practice. Nevertheless, for simplicity of explanation, the basic min-max timing simulation algorithm is first presented assuming the naive strategy for detecting transition orderings. In the next subsection, it is shown how more accurate results can be obtained by a sophisticated reconvergent fanout analysis technique.

Let q be a gate input of G different from p. Let the 13-valued waveform on q be $\langle b, m, e\rangle$. If the algorithm determines that q stabilizes to its final value before p starts transitioning, the final value of q, namely $\langle e, e, e\rangle$, is used to determine the potential sensitizability of G to p. Similarly, if the algorithm finds that q starts transitioning only after p has stabilized to its final value, the initial value of q, namely $\langle b, b, b\rangle$, is used to determine the potential sensitizability of G to p. However, if the algorithm fails to detect an ordering between the transitions on p and q, it uses the value $\langle b, m, e\rangle$ on q to determine the potential sensitizability of G to p. This represents the fact that q could be at its b or e values or transitioning in between when p transitions. This strategy ensures that the value of q used is always the "most-sensitizing-value" for determining the potential sensitizability of G to p. Specifically, G is never de-sensitized to a transition on p unless the algorithm detects an ordering between gate input transitions that causes G to be de-sensitized.

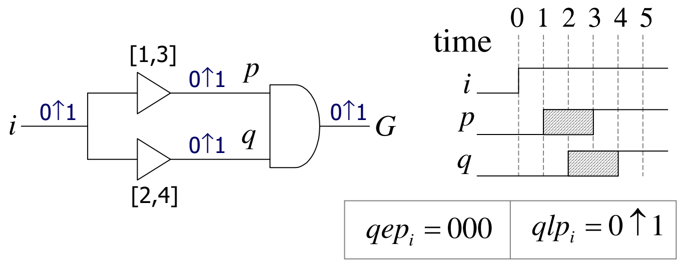
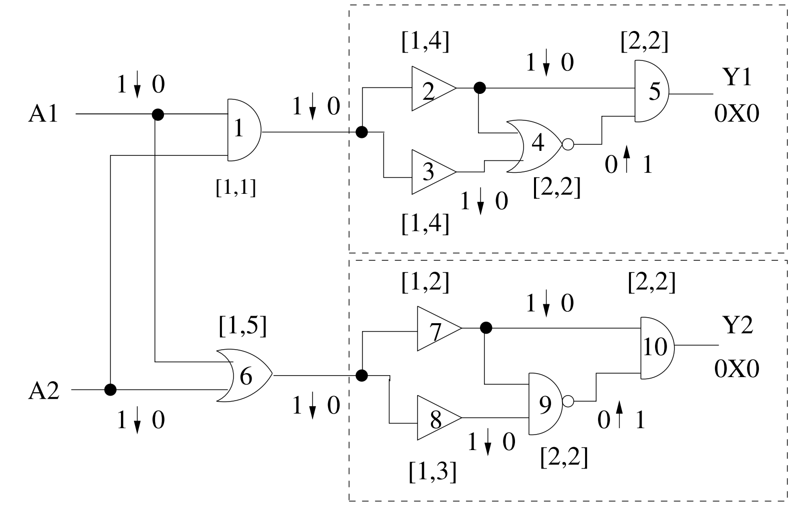


Figure 2.7: Computation of qep_i and qlp_i.

Fig. 2.6 shows the steps in function MinMaxAnalyze. To determine whether p is affected by primary input i (in steps 1 and 4), the algorithm checks if $p.del[i]$ is different from $(+\infty, -\infty)$. The qep_i and qlp_i terms in steps 1(b)(i) and 1(b)(ii) of Fig. 2.6 give the most sensitizing values of q at the earliest and latest times that a transition on primary input i reaches p. These are illustrated in Fig. 2.7. The test for potential hazards in step 2 is performed by checking whether the output of G is one of $\langle 0, X, 0\rangle$, $\langle 1, X, 1\rangle$, $\langle 0, X, 1\rangle$ or $\langle 1, X, 0\rangle$. To determine whether a potential hazard is masked in step 3, the ordering of gate input transitions is taken into account and the waveform at the output of G recomputed. If the m-component of the resulting waveform does not have the value an X, the potential hazard is said to be masked. For example, in Fig. 2.2, if the second input to the AND function falls before the first input rises, the output of the function is $\langle 0, 0, 0\rangle$, not $\langle 0, X, 0\rangle$ as would be computed without considering input orderings.

Fig. 2.8 shows the results of applying the min-max timing simulation algorithm to an example circuit adapted from [50]. It is assumed that the naive strategy for detecting ordering of transitions is used. Since the circuit is symmetric with respect to $A1$ and $A2$



GATE:	A1	1	2	3	4	5	6	7	8	9	10
DEL[A1]:	[0,0]	[1,1]	[2,5]	[2,5]	[4,7]	[4,9]	[1,5]	[2,7]	[2,8]	[4,10]	[4,12]

Gate delays are [min, max]. Wires have zero delay.

Figure 2.8: Basic min-max algorithm applied to example circuit from [50].

for the given input stimulus, the delay bounds from $A2$ to the internal gates are identical to those from $A1$. The results are more pessimistic than the reader might expect because multiple primary inputs affect the inputs of each gate. A careful manual analysis shows that the bounds computed for gates 5, 9 and 10 are conservative. Similarly, the 13-valued waveforms on the outputs of gates 5 and 10 are pessimistic: these gates actually have stable 0 at their outputs, whereas the analysis indicates potential static-0 hazards.

The following theorem characterizes the delay bounds computed by the min-max timing simulation algorithm.

Theorem 2.1 *If a transition on primary input i propagates to the output of gate G, the minimum propagation delay from i to G is bounded below by $G.del[i].t$ and the maximum propagation delay from i to G is bounded above by $G.del[i].T$.*

Proof: Without loss of generality, it is assumed that the gates are assigned topological indices such that all gates driving the inputs of gate G have topological indices less than that of G. The proof is by complete induction on the topological indices.

The following notation and terminology are used in the proof. A *controlling value* of a basic gate is an input value that determines the output value regardless of the values on the other gate inputs, e.g., a 0 input to an AND gate or NAND gate. A *non-controlling value* is a value that is not a controlling value for the gate. A gate input, as noted earlier, is an input to the zero-delay functional block of the gate delay model (see Fig. 2.3). All times are assumed to be relative to the time of transition of primary input i. The earliest time that a transition on i reaches p is denoted t_p, and the latest time that a transition on i reach p is denoted T_p.

Basis: Step 2(a) of the top-level algorithm (Fig. 2.5) ensures that the theorem applies to all primary inputs.

Hypothesis: Let the theorem apply to all gates with indices less than or equal to n.

Induction: Consider gate G (not a primary input) with index $n + 1$. Let a transition on primary input i propagate to the output of G. In the following, it is shown that $G.del[i].t$ is a lower bound on t_G; the proof that $G.del[i].T$ is an upper bound on T_G is similar and is omitted.

Let G' be a gate such that its output is connected to input p of G, and suppose the transition on i propagates through the output of G' in order to reach the output of G at time t_G. Since G' feeds G, its topological index is no greater than n. Therefore, by hypothesis, $G'.del[i].t \leq t_{G'}$. This implies that $G'.del[i].t + d_{p,G} \leq t_{G'} + d_{p,G}$. However, from step 4(a) in Fig. 2.6, $p.del[i].t$ is the same as the left hand side of this inequality, and since G' is connected to p, the value of t_p must be at least $t_{G'} + d_{p,G}$. Therefore, $p.del[i].t \leq t_p$. Also, by definition, G is potentially sensitized to p at time t_G and $t_p \leq t_G$. The last two inequalities imply $p.del[i].t \leq t_p \leq t_G$.

Now, there are two cases that need to be considered:

(A) If the algorithm finds that G is potentially sensitized to p at time $p.del[i].t$, then by definition, the qep_i values indicate the sensitizability of G to p. Therefore, by step 4(a) of function MinMaxAnalyze, $G.del[i].t \leq p.del[i].t \leq t_G$.

(B) If the algorithm finds that G is not potentially sensitized to p at time $p.del[i].t$, then primary input i affects one or more other inputs of G that control G at that time. Let r be the *controlling* gate input (or one of the controlling inputs in case of a tie) with the largest value of $r.del[i].t$ (not $+\infty$, which would indicate that r is not affected by i). Since G has no controlling inputs at time t_G, we must have $t_r \leq t_G$. By the induction hypothesis and by arguments similar to those presented above for gate input p, it is easy to see that $r.del[i].t \leq t_r$. The last two inequalities imply $r.del[i].t \leq t_G$. The proof is now completed by showing that $G.del[i].t \leq r.del[i].t$.

By our choice of r, when gate input r is considered in the outermost loop of function MinMaxAnalyze, the algorithm finds that all other controlling inputs of G have potentially transitioned to non-controlling values by time $r.del[i].t$. In addition, since $r.del[i].t \leq t_G$, no gate input q can settle to a final controlling value before $r.del[i].t$. Therefore, by the induction hypothesis, all gate inputs q that eventually settle to a final controlling value have $q.del[i].T \geq T_q \geq r.del[i].t$. The most-sensitizing values for r computed in step 1 of function MinMaxAnalyze therefore indicate that G is potentially sensitized to r at time $r.del[i].t$. By step 4 of MinMaxAnalyze, $G.del[i].t \leq r.del[i].t$. Since $r.del[i].t \leq t_G$, this proves the theorem.

■

2.4.3 Reconvergent Fanout Analysis

Reconvergent fanouts are signals that branch out from the output of one gate and converge back at the inputs of another gate. It is well-known that reconvergent fanouts are the cause of pessimism in conventional min-max timing simulation [12]. This subsection describes an efficient technique to conservatively detect ordering of signal transitions in the presence of reconvergent fanouts. This helps in improving the accuracy of min-max timing simulation.

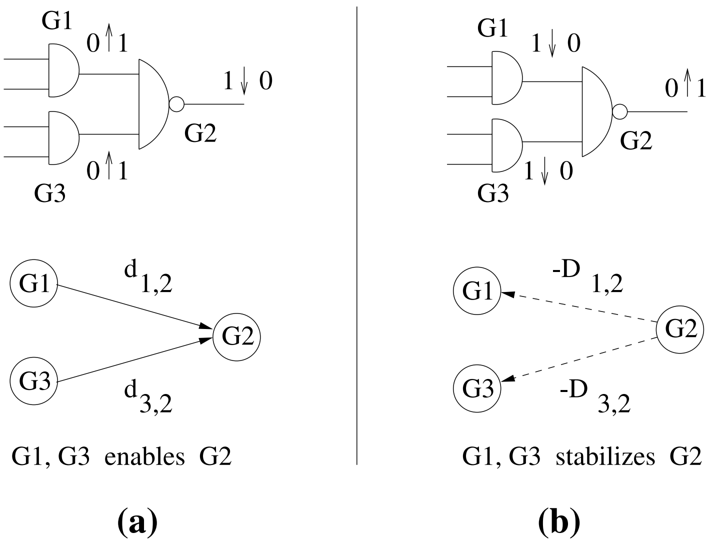


Figure 2.9: *enables* and *stabilizes* relations.

Given a combinational circuit and a set of primary input transitions, the following two relations are defined on the gates (including primary inputs):

- Gate G_1 *enables* gate G_2, if G_2 cannot transition until the transition on G_1 has propagated to one of its inputs. Fig. 2.9a shows an example. The enables relation can be represented by an *enabling graph*, where G_1 and G_2 are represented by vertices, and a solid directed edge is drawn from G_1 to G_2 (see Fig. 2.9a. If the minimum propagation delay from G_1 to G_2 is $d_{1,2}$, the edge is labeled with the weight $d_{1,2}$. Mathematically, if t_i denotes the time when G_i *starts transitioning*, $t_2 \geq t_1 + d_{1,2}$.

- G_1 *stabilizes* G_2, if G_2 can no longer transition after the last transition on G_1 has propagated to G_2. Fig. 2.9b shows an example of this relation. Here, the output of $G2$ cannot transition after the $\langle 1, \downarrow, 0\rangle$ transition on $G1$ reaches $G2$. Consequently, $G1$ stabilizes $G2$. This is graphically represented by a *stabilizing graph*, where a dotted directed edge is drawn from G_2 to G_1 (see Fig. 2.9b). If the maximum propagation delay from G_1 to G_2 is $D_{1,2}$, this edge is assigned the weight $-D_{1,2}$. Mathematically, if T_i denotes the time when G_i *ends transitioning*, then $T_2 \leq T_1 + D_{1,2}$.

The above relations depend on the particular input stimulus being simulated, which is implicit. [2]

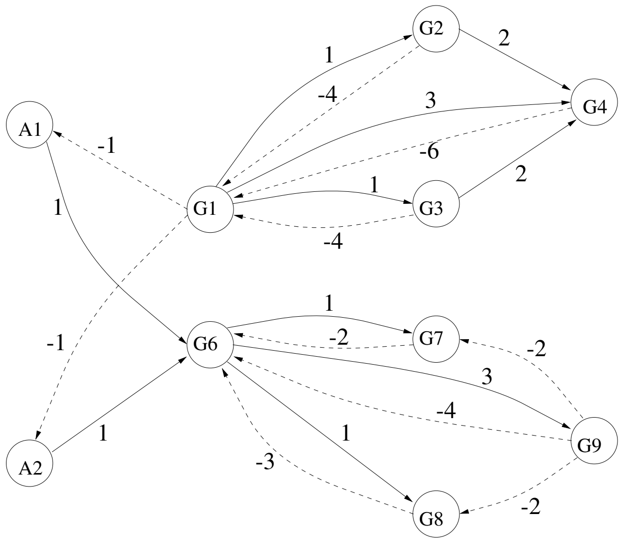


Figure 2.10: *enabling* and *stabilizing* DAGs for Fig. 2.8.

Each of the above relations defines a partial order on the gates. The corresponding graphs are, therefore, directed acyclic graphs (DAGs). Fig. 2.10 shows these DAGs for the circuit and primary input stimulus of Fig. 2.8, superimposed on each other. The vertex corresponding to gate i has been labeled G_i. For clarity, some of the edges that are implied by transitivity are not shown.

[2]Note that the *enables* and *stabilizes* relations were called *waits_for* and *yields_to* in a preliminary work [20].

Now consider the following sequence of edges in Fig. 2.10: $G7 \stackrel{-2}{\rightarrow} G6 \stackrel{3}{\rightarrow} G9$. The information represented by these edges is: $T_6 \geq T_7 - 2; \;\; t_9 \geq t_6 + 3$. This implies that $t_9 - T_7 \geq 1 + (t_6 - T_6)$. However, we know from the 13-valued annotations in Fig. 2.8) that each of $G6$, $G7$ and $G9$ has a hazard-free transition on it. Consequently, both t_6 and T_6 represent the transition time of $G6$, and similarly for $G9$ and $G7$. It then follows that $t_9 - T_7 \geq 1$, i.e., $G9$ transitions at least 1 time unit after $G7$. This implies that the hazard on $Y2$ in Fig. 2.8 does not occur.

The method for detecting transition orderings outlined above can be easily generalized. The enabling and stabilizing DAGs for a given circuit and primary input stimulus are first superimposed. A *directed path* in the resulting graph is a sequence of vertices $(G_1, G_2, \ldots G_n)$ such that an enabling or stabilizing edge exists from G_i to G_{i+1} for all i in 1 through $n-1$. Gate G_n transitions after G_1 if the directed path satisfies the following constraints:

1. Each gate G_r (other than G_1 and G_n) in the path has a clean (hazard-free) transition on it.

2. The summation of all edge weights along the path is positive.

3. If G_1 has a potential hazard on it, the first edge in the path, $\langle G_1, G_2 \rangle$, is a stabilizing edge.

4. If G_n has a potential hazard on it, the last edge in the path, $\langle G_{n-1}, G_n \rangle$, is an enabling edge.

This result is formalized in Theorem 2.2.

Theorem 2.2 *If there exists a directed path from G_1 to G_n satisfying the above constraints, then G_n starts transitioning after G_1 has stabilized to its final value.*

Proof: Consider a directed path $(G_1, G_2, \ldots G_n)$ satisfying the above constraints. It is assumed that the i^{th} edge $\langle G_i, G_{i+1} \rangle$ has weight w_i. Note that w_i could be positive or negative depending on the relation represented by $\langle G_i, G_{i+1} \rangle$.

First, consider the case where the first edge $\langle G_1, G_2 \rangle$ is a stabilizing edge and the final edge $\langle G_{n-1}, G_n \rangle$ is an enabling edge, as shown in Fig. 2.11. The constraints represented

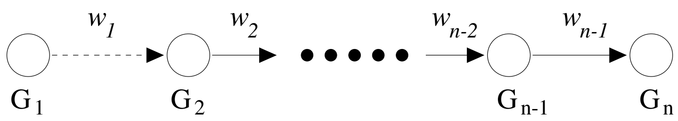


Figure 2.11: A path in the superimposed DAGs.

by this path can be expressed by the following set of inequalities.

$$\begin{aligned} t_2 = T_2 &\geq T_1 + w_1 \\ t_3 &\geq t_2 + w_2 \\ &\vdots \\ t_{n-1} &\geq t_{n-2} + w_{n-2} \\ t_n &\geq t_{n-1} + w_{n-1}. \end{aligned} \tag{2.1}$$

Note that since all intermediate gates have clean transitions (required by constraint 1), t_i is the same as T_i for all gates G_i other than G_1 and G_n. Therefore, the inequalities in (2.1) hold regardless of the type (enabling/stabilizing) of the intermediate edges. It follows from inequalities (2.1) that

$$t_n - T_1 \geq \sum_{i=i}^{n-1} w_i = \sigma. \tag{2.2}$$

Therefore, if σ is positive, we have $t_n > T_1$. In other words, G_n starts transitioning only after G_1 has stabilized to its final value.

Now, consider the case where $\langle G_1, G_2 \rangle$ is an enabling edge. Constraint 3 requires that G_1 must have a clean transition. This implies that t_1 is the same as T_1 and the set of inequalities (2.1) still apply. Similarly, if the final edge $\langle G_{n-1}, G_n \rangle$ is a stabilizing edge, constraint 4 requires that G_n must have a clean transition. Therefore, t_n is the same as T_n and inequalities (2.1) apply. Hence, the theorem is proved. ■

The above theorem guarantees that the technique described above never reports a false ordering of transitions. Note, however, that the technique is conservative and may fail to report an ordering of transitions that would have been detected by a more expensive and detailed analysis.

When constructing a path in the superimposed DAGs in accordance with constraints 1, 3 and 4, if a cycle is formed, the length of this cycle must be non-positive. To see why this is so, note that constraint 1 implies that at most one gate in a cycle can have a hazard or multiple transitions on its output. Moreover, if there is one such gate, G, it must be the first as well as the final gate in the cycle under consideration, since all intermediate gates must have clean transitions. In other words, both G_1 and G_n refer to gate G. Constraint 3 now requires that the edge coming out of G must be a stabilizing edge and constraint 4 requires that the edge coming into G must be an enabling edge. Given these edges, it can be easily shown (using the same technique used to derive inequality (2.2)) that the length of the cycle, σ, is bounded above by the difference $(t_G - T_G)$. Since T_G is always at least as large as t_G, it follows that σ is bounded above by 0. However, if the cycle has no gates with hazards, then for any gate G' in the cycle, $t_{G'}$ is the same as $T_{G'}$. Depending on whether the edges coming into and going out of G' are enabling or stabilizing edges, we have one of the following inequalities:

$$t_{G'} - T_{G'} \geq \sigma \ \text{ or } \ T_{G'} - t_{G'} \geq \sigma \ \text{ or } \ t_{G'} - t_{G'} \geq \sigma \ \text{ or } \ T_{G'} - T_{G'} \geq \sigma.$$

Since $t_{G'}$ equals $T_{G'}$ in each case, σ is equal to 0. Therefore, the length of the cycle is always non-positive.

Since cycles are of non-positive length, the problem of finding a path of positive length from vertex X to vertex Y in a graph with V vertices and E edges can be solved in $O(V.E)$ time using the Bellman-Ford longest path algorithm [28] with the additional constraints 1, 3 and 4. In fact, since the lengths of paths between multiple pairs of vertices may be needed when applying this technique to improve the accuracy of min-max timing simulation, a longest-path variant of Floyd-Warshall's algorithm [28] may be used to compute the longest path lengths between all pairs of vertices in $O(V^3)$ time. The results may then be stored and subsequently used by the min-max timing simulation algorithm to determine if pairs of transitions are ordered in time. Note, however, that an algorithm that computes an underestimate of the true longest path would also suffice for our purpose, as long as the underestimate is > 0. Such an underestimate can often be efficiently computed by considering only a subset of all the paths between vertices X and Y.

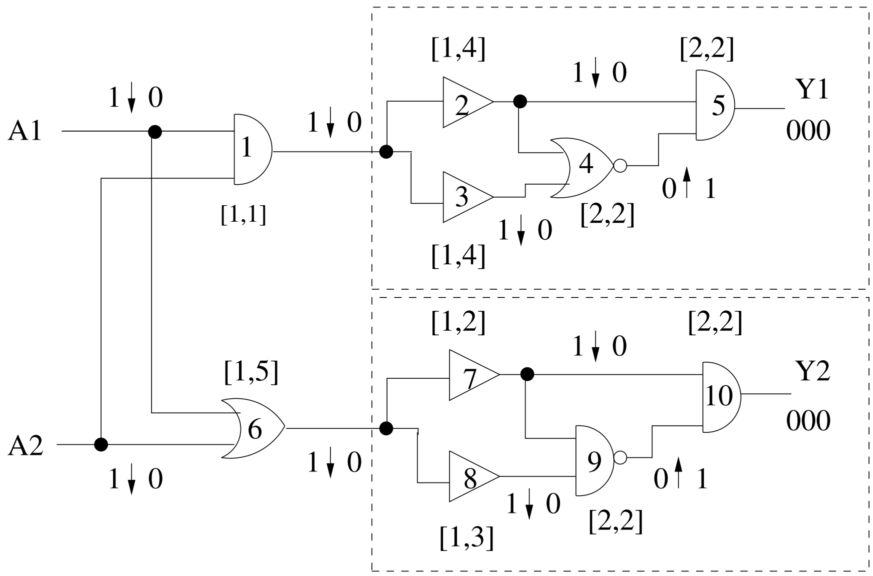
Computing the enabling and stabilizing DAGs for a given circuit and input stimulus is straightforward. Gates are processed in topological order. Let G be a gate with a potential transition on its output and let p be a transitioning input of G. To determine whether p enables G, input p is held at its initial state ($\langle b, b, b\rangle$) and all other transitioning inputs of G are allowed to transition. If 13-valued evaluation now indicates that G *does not* have a potential transition on its output, p is said to enable G. There also exists another situation that often arises in practice. Suppose the set of transitioning inputs of G is $\{p_i, \ldots p_n\}$. If there exists a gate G' which enables each p_k in the set $\{p_i, \ldots p_n\}$, then G' also enables G. The rationale behind this is that a stable gate must be enabled by at least one of its inputs for it to transition, and each transitioning input, in turn, is enabled by G'. Once these simple relations have been determined, the transitive property can be used to add new edges in the enabling DAG. When computing edge weights in the enabling DAG, if there exists a sequence of edges $G'' \xrightarrow{a} G' \xrightarrow{b} G$ and if the current weight of edge $G'' \to G$ is c, the new weight of $G'' \to G$ is easily seen to be $\max(c, a + b)$. The procedure for computing the stabilizing DAG is similar. Specifically, to determine if gate input p stabilizes G, input p is held at its final state ($\langle e, e, e\rangle$) and all other gate inputs are allowed to transition. If 13-valued evaluation indicates that G does not have a potential transition on its output, then p is said to stabilize G. In addition, if there exists a gate G' that stabilizes all the transitioning inputs of G, then G' also stabilizes G. Finally, if there exists a sequence of edges $G'' \xrightarrow{a} G' \xrightarrow{b} G$ in the stabilizing DAG and if the current weight of edge $G'' \to G$ is c, the new weight of $G'' \to G$ is $\max(c, a + b)$. It is easy to see that for a circuit with n_g gates, each DAG can be constructed in $O(n_g^3)$ time by first constructing the edges as indicated above, and then invoking the transitive closure algorithm [28] to compute the transitive closure of each DAG.

The reconvergent fanout analysis technique described above has been used in the basic min-max timing simulation algorithm of Section 2.4.2 to determine more accurately whether a gate is *de-sensitized* to one of its inputs. By Theorem 2.2, the analysis never reports a false ordering of transitions, so gates are never falsely de-sensitized. Specifically, whenever a gate is found to be de-sensitized to one of its inputs due an ordering of input transitions, the gate is indeed de-sensitized. Consequently, delay bounds computed by the min-max algorithm with reconvergent fanout analysis are still conservative, though often more accurate than before.

If the enabling and stabilizing DAGs are constructed and analyzed once using the all-pairs shortest paths algorithm and the results stored and subsequently utilized during min-max timing simulation, the complexity of the min-max timing simulation algorithm with reconvergent fanout analysis is seen to be $O(n_g^3 + n_g.n_{pi}.n_{fanin}^2)$, where n_g is the number of gates, n_{pi} is the number of primary inputs and n_{fanin} is the maximum fanin of a gate. In the current implementation of the algorithm, for efficiency, the enabling and stabilizing DAGs are only partially constructed, and underestimates of longest path lengths are used. This results in a more conservative analysis which has complexity $O(n_g.n_{pi}.n_{fanin}^2.n_g^*)$, where n_g^* is the number of incoming enabling edges or outgoing stabilizing edges at a gate. The worst case value of $n_g^* = n_g$; in practice, however, it is a small number, typically 10 or less. For technological reasons, n_{fanin} is also typically 8 or less. Consequently, the observed complexity is $O(n_g.n_{pi})$.

Fig. 2.12 shows the results of applying the min-max timing simulation algorithm with reconvergent fanout analysis to the circuit of Fig. 2.8. It may be verified that all the computed delay bounds and 13-valued waveforms are now exact.

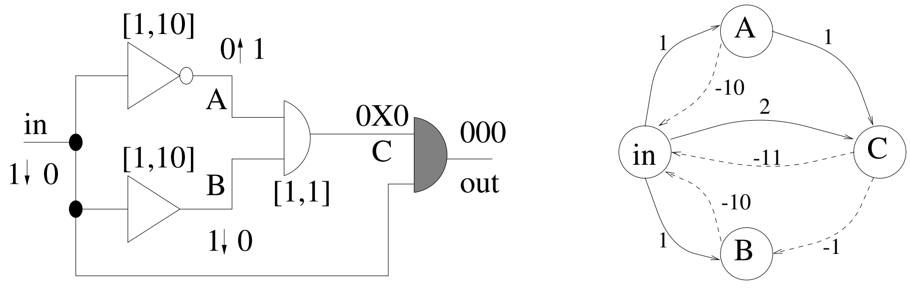
Fig. 2.13 shows another example of the proposed algorithm at work. The circuit in Fig. 2.13a is adapted from Lindermann's thesis [64], where it was shown that MTV conservatively predicts a hazard at the output of the circuit, although the output actually remains stable at 0. Using the proposed reconvergent fanout analysis technique, the DAGs shown in Fig. 2.13b are obtained. Observe that there exists an enabling edge from in to C because each of the two inputs, A and B, of C are enabled by in. Consequently, C is also enabled by in. Since there exists a path of positive weight from in to C, the proposed algorithm detects that C transitions after in. Therefore, the min-max timing simulator finds that the shaded AND gate is de-sensitized to the transition on C, so $\langle 0, 0, 0\rangle$ is produced at the output of the circuit.



GATE:	A1	1	2	3	4	6	7	8	9
DEL[A1]:	[0,0]	[1,1]	[2,5]	[2,5]	[4,7]	[1,5]	[2,7]	[2,8]	[4,9]

Gate delays are [min, max]. Wires have zero delay.

Figure 2.12: More accurate simulation with reconvergent fanout analysis.



(a) (b)

Gate delays are [min, max]. Wires have zero delay.

Figure 2.13: Event ordering not detected by MTV [64].

2.5 Pathological Examples

This section describes two pathological examples where the proposed algorithm fails to compute exact results. The existence of such examples is not surprising since the proposed algorithm is conservative in the worst case. However, it is conjectured that such cases seldom arise in real life, and cannot be analyzed in an efficient manner.

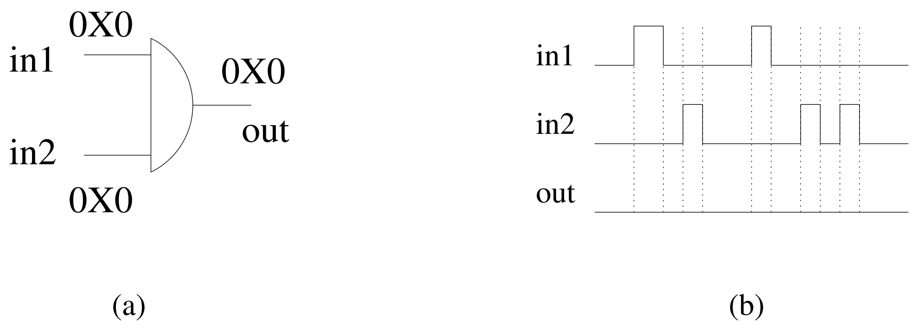
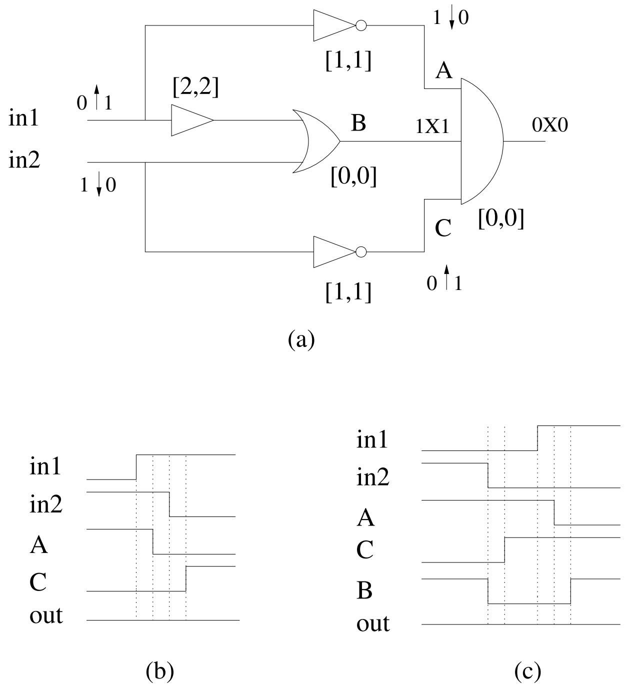


Figure 2.14: First pathological example.

Fig. 2.14 shows the first example. It is assumed that a large number of interleaved transitions are generated at the inputs of an AND gate deep inside a larger circuit, in response to a stimulus applied at the primary inputs of the circuit. The timing of the interleaved transitions are such that the gate output remains stable at 0. However, the proposed algorithm abstracts each sequence of transitions using X's, and discards information about the timing of intermediate transitions. Consequently, it finds two overlapping X transitions on the gate inputs and conservatively predicts an X at the output of the gate. Note that in order to obtain the exact output waveform in this example, one must remember the timing of all intermediate transitions. Since the number of transitions can be exponential in the size of the circuit in the worst case, this would require an exponential time algorithm.

Fig. 2.15 shows another pathological example. Here, the transitions on A and C are not ordered – A can transition before C and vice versa, as shown in Figs. 2.15b and c. However, an inspection of Figs. 2.15b and c indicates that regardless of the order in which A and C transition, the output of the circuit remains stable at 0. Since the proposed reconvergent fanout analysis technique is able to detect an ordering of two input transitions only



Gate delays are [min, max]. Wires have zero delay.

Figure 2.15: Second pathological example.

if they are ordered for all input transition times, no ordering of transitions on A, B and C are reported. Consequently, the min-max timing simulator conservatively infers that the transitions could overlap to produce a hazard at the output. Note that in order to determine the exact waveform at the output of this circuit, one needs to analyze the circuit for all possible orderings of input transitions.

2.6 Experimental Results

The min-max timing simulation algorithm described above is primarily meant for use in analyzing fundamental-mode asynchronous circuits. Towards this end, a timing analysis tool for a class of fundamental-mode circuits, called 3D circuits [99], has been designed using the algorithm described above. Structurally, each 3D circuit is a combinational network with some of its outputs fed back as inputs to the network. In order to analyze a 3D circuit, the feedback loops are cut and the resulting combinational network simulated after applying appropriate input stimuli. Detailed results obtained by applying the tool to a suite of 3D benchmarks are presented in Chapter 3. In this section, a subset of the results that demonstrate the efficiency and effectiveness of the combinational min-max timing simulation algorithm is presented.

Table 2.1 lists a set of 3D benchmark circuits and gives the number of gates and primary inputs of the combinational networks obtained after cutting the feedback loops of each 3D circuit. Note that after the feedback loops are cut, the primary inputs and feedback inputs of the original 3D circuit become the primary inputs of the combinational circuit. For each combinational circuit thus obtained, the number of input stimuli applied is also listed. An input stimulus is a set of 13-valued waveforms, one waveform for each primary input of the combinational circuit. For each input stimulus, the algorithm computes bounds on the signal propagation delays from *each* transitioning primary input to each gate. For example, in the case of "pscsi", the algorithm computes signal propagation delays from each transitioning primary input to each internal gate 225 times, once for each input stimulus. The column labeled "Analysis Time" gives the total time taken to simulate all the input stimuli. The CPU times shown are on a DEC 5000/240 machine, and do not include the time required to read in the circuit description, and topologically sort the gates, which are negligible.

The nominal gate delays are estimated using a Hitachi CMOS gate library [1]. Each gate in the library has a nominal delay given by: $d = d_0 + K.C$, where C represents the output capacitance and K is a gate-specific parameter. The value of C is computed as $0.4 \cdot E$ (obtained from [1]), where E is the *effective fanout* of the gate. The effective fanout is obtained by summing the *normalized load* presented by all gates driven by the fanouts. The values of d_0, K and normalized load for gates are obtained from the Hitachi data book [1], and are different for rising and falling transitions. The gate delays used in the experiments are assumed to vary within $(0.9d, 1.1d)$, where d is the nominal gate delay computed as above. The percentage variation in nominal delays is actually an adjustible parameter of the tool that can be modified to analyze the effects of other gate delay variations. Wire delays are assumed to be zero delays since post-layout information about wire delays were not available.

In order to evaluate the improvement in accuracy due to reconvergent fanout analysis, several 3D circuits were also analyzed with reconvergent fanout analysis turned off. Table 2.2 shows the results of these experiments on some of the circuits in which there were significant differences in results. In this table, two types of timing constraints of 3D circuits, namely hold-time (HT) and fundamental-mode (FM) constraints, are reported. Details of these constraints are explained in Chapter 3; nevertheless, *a larger value of a constraint indicates a more conservative analysis*. The columns titled "-b" represent results obtained with the basic algorithm, i.e., without reconvergent fanout analysis, whereas columns titled "-r" give results obtained using the proposed reconvergent fanout analysis. The bold-faced entries indicate that reconvergent fanout analysis results in significantly less conservative constraints than those obtained with the basic algorithm. For the 3D benchmarks in Table 2.2, the figures in the "-r" column also give *exact* timing constraints for the circuits under consideration. This indicates that the bounds computed by the min-max timing simulator are fairly accurate in practice.

While the results presented in this section are promising, the accuracy and efficiency of the proposed algorithm need to be investigated with more extensive experimentation. Unfortunately, the tools described in [31, 52, 51] were unavailable for performance and accuracy comparisons. Therefore, comparison results with exact min-max timing simulators

3D Benchmark	#Primary inputs of combinational circuit	#Gates	#Stimuli	Analy-sis Time(s)
ircv	14	107	100	1.375
ircv-bm	12	58	45	0.309
ircv-csm	12	64	45	0.339
isend	16	247	146	7.738
isend-bm	13	104	68	1.052
isend-csm	12	81	68	0.698
trcv	14	96	100	1.318
trcv-bm	12	58	45	0.332
trcv-csm	12	59	45	0.328
tsend	16	182	140	4.761
tsend-bm	12	65	65	0.466
tsend-csm	13	69	65	0.492
biu-dma2fifo	12	81	67	0.711
biu-fifo2dma	13	77	52	0.590
scsi-init-send	9	33	32	0.105
scsi-targ-send	9	41	32	0.155
pscsi	22	350	225	17.171

Table 2.1: CPU times for min-max timing analysis.

3D Benchmark	HT-b	HT-r	FM-b	FM-r
ircv	7.140	7.140	**4.998**	**4.789**
ircv-bm	**5.130**	**4.488**	**2.995**	**1.959**
trcv	**6.525**	**5.594**	**3.201**	**2.715**
trcv-bm	**5.637**	**5.501**	2.073	2.073
tsend-bm	**5.915**	**5.860**	**2.737**	**2.600**
biu-dma2fifo	7.390	7.390	**3.402**	**2.550**
biu-fifo2dma	4.753	4.753	**2.074**	**1.357**
scsi-targ-send	4.172	4.172	**0.628**	**0.540**

Columns with “-b” give results without reconvergent fanout analysis; those with “-r” give results with reconvergent fanout analysis. All constraints are in terms of nominal inverter delay with fanout 4.

Table 2.2: Illustrating usefulness of reconvergent fanout analysis.

could not be obtained.

2.7 Summary

Min-max timing simulation is an important technique for analyzing the temporal behavior of circuits with bounded component delays. This chapter described a polynomial-time algorithm for approximate min-max timing simulation of combinational circuits. An efficient reconvergent fanout analysis technique for increasing the accuracy of simulation was also described. The intended application of the algorithm is timing analysis of fundamental-mode asynchronous circuits. Experimental results with a suite of asynchronous benchmarks indicate that (a) the algorithm is efficient even when simulating a large number of input stimuli, and (b) reconvergent fanout analysis helps in obtaining more accurate results.

It is likely that the proposed algorithm can be used for analyzing certain non-fundamental-mode circuits as well. For example, Lavagno, Keutzer and Vincentelli [60] have described a procedure for enforcing ordering of signal transitions by inserting delay buffers in asynchronous circuits synthesized from Signal Transition Graphs (STG). Their method uses bounds on signal propagation delays from the circuit inputs to internal gates to estimate the required buffer delays. However, they do not address the problem of efficiently computing the signal propagation delays in a general setup. The algorithm described in this chapter can potentially be used to obtain these delay bounds. However, a discussion of this is beyond the scope of this work.

Chapter 3

Timing Analysis of Extended Burst-Mode Circuits

3.1 Introduction

This chapter describes an interesting application of the min-max timing simulation algorithm described in Chapter 2. Specifically, a timing analysis tool for a class of asynchronous circuits, called *extended burst-mode* circuits [105, 103, 99] implemented in the *3D* design style [104, 102, 99] (a practical asynchronous design style), is described. Given a gate-level 3D circuit with bounded gate and wire delays, the tool uses the min-max timing simulation algorithm of Chapter 2 to check timing constraints required for correct operation of the circuit. It then reports usage constraints and timing information sufficient to ensure correct operation of the circuit. Although the results computed by the tool represent conservative approximations to the true timing requirements in the worst case, experiments indicate that the analysis is fairly accurate and efficient in practice.

A 3D asynchronous circuit can be viewed as a combinational network with some (or all) of its outputs fed back as inputs to the network [99]. 3D design [104, 102, 99] guarantees that the combinational circuit obtained by cutting the feedback loops is free of *logic and functional hazards* [89, 71] regardless of the delays of gates and wires. However, there are certain global timing constraints that must be satisfied when the sequential behavior of the circuit is considered. These constraints depend on the delays of gates and wires in the circuit implementation. Since variations in manufacturing and operating conditions result in uncertainties in component delays in a chip, it is important to find bounds on usage constraints and other timing information that are sufficient to ensure the correct operation of the circuit (henceforth, these are called *safe* timing constraints).

In this chapter, *3D design is assumed to be bug-free*. In other words, a 3D circuit is assumed to operate correctly if all global timing constraints are satisfied. Therefore, hazards or spurious transitions at the outputs of the circuit are generated solely due to timing constraint violations.

The first step of the analysis method is to cut the feedback loops of the 3D circuit to produce a combinational circuit. The combinational circuit has the primary inputs and outputs of the original 3D circuit, and also inputs and outputs from the feedback loops that were cut. The combinational circuit is now analyzed as follows.

(a) As the 3D asynchronous machine transitions from one state to another, the signal transitions that appear at the primary inputs of the combinational circuit are determined. This is accomplished by analyzing the behavioral specification of the 3D asynchronous machine, and by copying transitions from the feedback outputs of the 3D circuit to the corresponding feedback inputs.

(b) For each set of input transitions, gates with spurious transitions or hazards on their outputs generated as a result of timing constraint violations are then identified. In this chapter, these gates are called problem gates.

(c) Next, bounds on the signal propagation delays *from each circuit input* to each problem gate is determined using the min-max timing simulation algorithm of Chapter 2.

(d) Finally, the signal propagation delay bounds are used to derive usage constraints that ensure correct operation of the circuit.

The idea outlined above has been used to design a completely automated timing analysis tool for 3D circuits. Experiments indicate that the tool is capable of analyzing moderately large 3D circuits reasonably efficiently and with fairly good accuracy.

The remainder of this chapter is organized as follows. Section 3.2 briefly reviews extended burst-mode specifications and the 3D design style. Section 3.3 describes useful properties of 3D circuits that facilitate the design of a simple yet effective timing analysis tool. Section 3.4 describes how problem gates in 3D circuits can be efficiently identified using 13-valued simulation, explained in Chapter 2. Section 3.5 explains how timing constraints for correct operation of 3D circuits can be determined using min-max timing simulation. Section 3.6 describes an automatic timing analysis tool for 3D circuits and reports results of applying the tool to a suite of 3D benchmarks. Finally, the contributions of the chapter are summarized in Section 3.8.

3.2 Extended Burst-Mode and 3D Design: A Review

This section briefly reviews the extended burst-mode specification style for asynchronous circuits, and highlights the main characteristics of 3D designs – a practical implementation style for extended burst-mode specifications.

3.2.1 Extended Burst-Mode Specifications

Extended burst-mode [105, 103, 99] is a powerful specification formalism that can be used for specifying a large and useful class of asynchronous state machines, synchronous state machines, and systems with both synchronous and asynchronous components. This section reviews some definitions and notations from [99] used later in the chapter.

Fig. 3.1 shows an example extended burst-mode specification of a Bus Interface Unit that interfaces between a DMA bus and FIFO buffer of a SCSI controller. Details of the specification are described in Yun's thesis [99]. Signals enclosed within angle brackets, like $\langle \mathsf{cntgt1}+\rangle$, represent *conditional* signals. Specifically, $\langle \mathsf{cntgt1}+\rangle$ and $\langle \mathsf{cntgt1}-\rangle$ denote the clauses "if $\mathsf{cntgt1}$ is 1" and "if $\mathsf{cntgt1}$ is 0". Signals which are not enclosed within angle brackets are called *edge* signals. An edge signal with a $+$ or a $-$ (e.g., $\mathsf{ok}+$ in Fig. 3.1) constitutes a *terminating* edge, while one with a $*$ (e.g., $\mathsf{frin}*$) is a *directed don't care*. A terminating edge $\mathsf{ok}+$ denotes a $0 \rightarrow 1$ transition of ok if ok was initially 0, and no transition if ok was initially 1. The interpretation of $\mathsf{ok}-$ is similar. A directed don't care is a signal that can change exactly once during a multi-state path in the state transition diagram (it is a "don't care" because the specification does not care *where* along the path

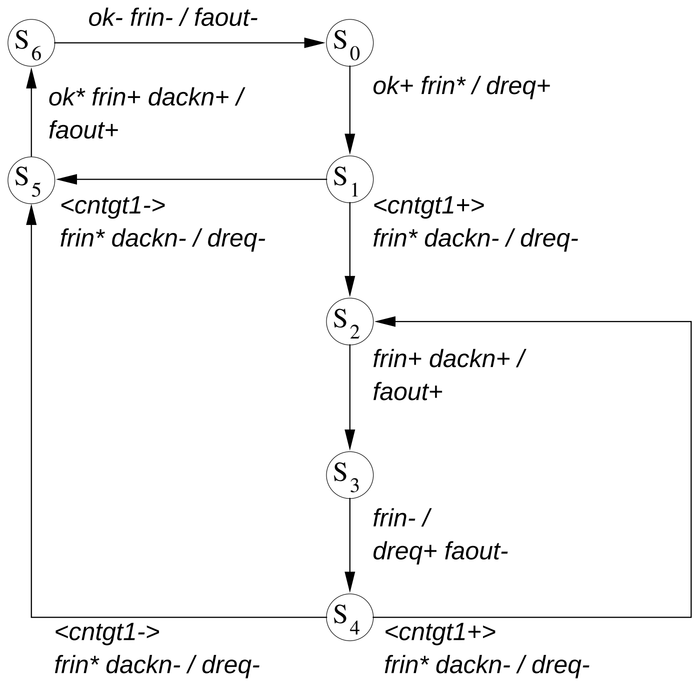


Figure 3.1: Example of extended burst-mode specification.

the transition occurs). For example, in Fig. 3.1, frin is low in state S_0. If we consider the sequence of state transitions $S_0 \to S_1 \to S_5 \to S_6$, then frin can rise exactly once during this sequence, and must have risen by the time the machine reaches state S_6. If there exists a terminating edge that is not immediately preceded by a directed don't care (e.g., dackn+ during the state transition $S_2 \to S_3$ preceded by dackn− during $S_1 \to S_2$ in Fig. 3.1), the terminating edge is called a *compulsory* edge. If a conditional signal is not specified in a state transition, it may transition freely during the state transition. Edge signals must, however, remain stable if they do not participate in a state transition.

An *input burst* consists of a non-empty set of input edges, at least one of which must be compulsory. The edges may appear in any order. An *output burst* consists of a possibly empty set of output edges. The output transitions may also be generated in any order. When the machine is in a given state, and all conditional signals are stable at their desired values, the arrivals of all terminating edges in an input burst causes the corresponding output burst to be generated and the machine transitions to the specified next state. For example, the behavior of the machine in state S_1 in Fig. 3.1 is as follows: if cntgt1 is 1 when dackn falls, then the machine transitions to state S_2 and output dreq falls; if cntgt1 is 0 when dackn falls, the machine transitions to state S_5 and output dreq falls.

3.2.2 3D Design

The 3D design style, developed by Yun, Dill and Nowick [104, 102, 103, 99], is currently the most practical implementation method for extended burst-mode circuits. Formally, a 3D machine is defined as a 4-tuple (X, Y, A, δ), where X is a set of primary input symbols, Y a set of primary output symbols, Z a possibly empty set of state variable symbols, and $\delta : X \times Y \times Z \to Y \times Z$ is a *next-state function*. The next-state function δ is typically represented by a 3-dimensional next-state table (hence the name 3D). The hardware implementation of a 3D machine consists of a combinational network implementing the next state table, with some of its outputs fed back as inputs to the network. There are no explicit storage elements, such as latches or C-elements, and the machine's state is stored in the feedback wires. The synthesis method developed in [99] guarantees that if the feedback paths are opened, the combinational logic implementing the next state function is free of function and logic hazards, regardless of gate and wire delays.

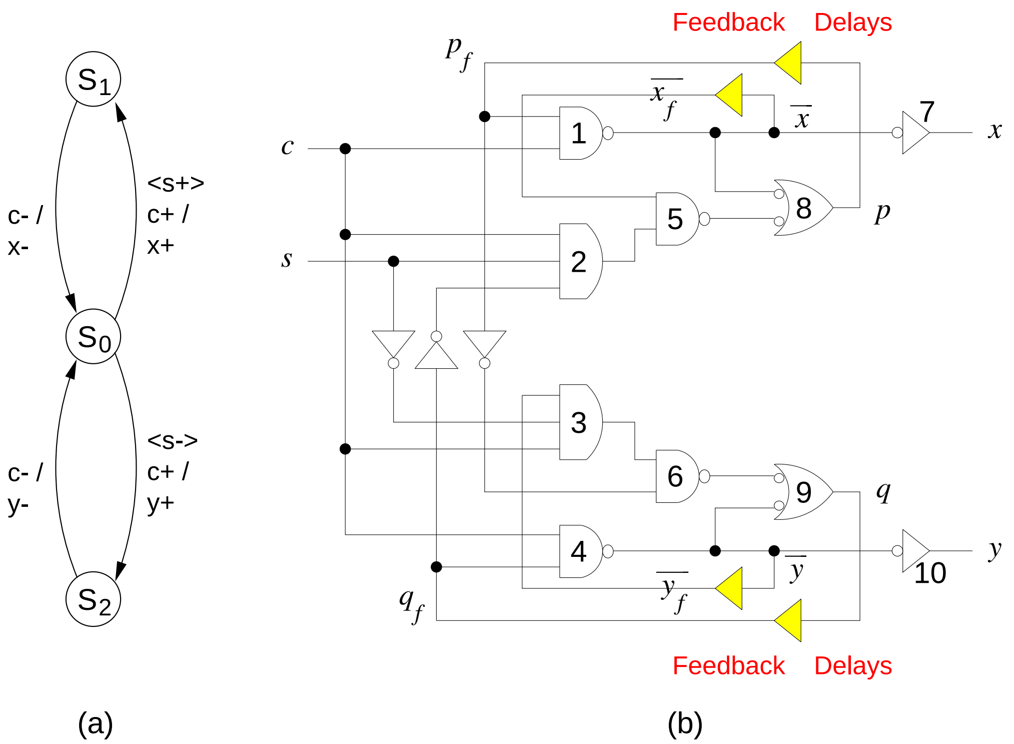


Figure 3.2: (a) Extended burst-mode specification (b) 3D implementation.

Fig. 3.2 shows a simple extended burst-mode specification and its 3D implementation. The behavior of this circuit can be summarized as follows: if the mode bit s is 1 when it is sampled by the rising edge of signal c, then output x follows c for one cycle while output y remains 0. If the sampled value of s is 0, y follows c and x remains 0.

Given an input burst, the response of a 3D machine is comprised of two or three phases, depending on the circuit implementation (see Fig. 3.3). During the first phase ($\phi 1$), transitions applied to the primary inputs propagate forward through the combinational network and give rise to transitions on the primary outputs and/or feedback signals. These are then fed back as inputs to the combinational network, and the second phase of operation ($\phi 2$) begins. During this phase, the fed back transitions propagate forward through the combinational network. If there are only two phases of operation, all signals in the circuit stabilize

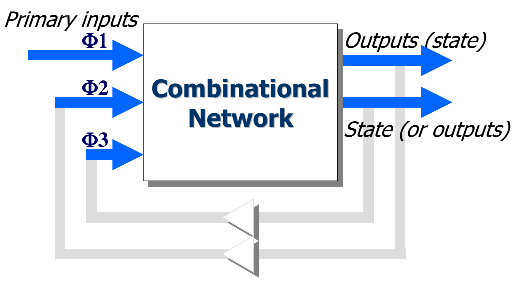


Figure 3.3: Phases in the operation of a 3D circuit.

at the end of the second phase. Otherwise, transitions on some primary outputs or feedback signals are produced at the end of the second phase. These are again fed back as inputs to the circuit, and the third phase of operation ($\phi 3$) follows. All signals in the circuit stabilize at the end of the third phase.

There are certain global timing constraints when the sequential behavior of a 3D circuit is considered. These are summarized below:

(a) *Fundamental-mode requirement*: A minimum time interval must elapse between the last primary output transition of the current burst and the first compulsory input transition of the next burst. This is the classical *fundamental-mode* environmental constraint of asynchronous circuits [71, 89].

(b) *Minimum feedback delay requirement*: The feedback paths should have a certain minimum delay in order to avoid *essential hazards* [89] – a type of sequential hazards specific to asynchronous circuits.

(c) *Setup and hold-time requirement*: Each conditional signal that is sampled by an input burst must remain stable from a certain *setup-time* before the first compulsory transition to a certain *hold-time* after the last terminating transition in the input burst.

3.3 Useful Properties of 3D circuits

Functionally correct 3D circuits enjoy special properties which simplify their timing analysis. These are summarized below:

- Any hazard appearing on the outputs of the combinational circuit during the first phase is due to setup-time violations.
- Assuming there are no setup-time violations, any hazard appearing on the outputs of the combinational circuit during the second and third phases is either due to insufficient feedback delays or due to hold-time violations.
- Assuming there are no setup-time, feedback delay or hold-time constraint violations, any hazard appearing on the outputs of the combinational circuit when two consecutive input bursts are applied is due to a fundamental-mode constraint violation.

Because of these properties, it is desirable to analyze the combinational part of a 3D circuit for each phase of each extended burst-mode state transition separately. This effectively isolates the effects of the different types of timing constraint violations from each other, simplifying the analysis of each potential violation.

Another important property of 3D circuits is the *absence of cyclic dependencies between the timing constraints*. This greatly simplifies timing analysis because it eliminates the need to worry about convergence when determining safe bounds on the timing constraints for correct operation of the circuit. The values of hold-time and fundamental-mode constraints depend on (i) differences in signal propagation delays along different paths in the combinational circuit, *and* (ii) delays in the feedback paths. However, the minimum feedback delays, as well as setup-time constraints are determined solely by the difference in signal propagation delays along different paths in the combinational circuit. Consequently, the feedback delays and setup-time constraints may be determined in any order, without any knowledge of the values of the other constraints. Once the delays in the feedback paths are known, they can be used to determine the hold-time and fundamental-mode constraints.

3.4 Identifying Problem Gates

A *problem gate* is one which has a hazard or spurious transition on its output, generated as a result of a timing constraint violation. Note that a 3D circuit may have gates with hazards on their outputs even if all timing constraints are satisfied. These are not called problem gates, because they do not cause a problem – 3D design guarantees that the hazards on the outputs of such gates are masked before reaching the outputs of the combinational circuit.

To identify problem gates efficiently, each phase of operation of each extended burst-mode state transition is simulated using the 13-valued abstract waveform algebra described in Section 2.3. The combinational circuit obtained by cutting the feedback loops is simulated under the following two conditions.

1. First, it is assumed that all global timing constraints are satisfied. Accordingly, appropriate 13-valued stimuli are applied to the inputs of the combinational circuit and the circuit simulated. All gates with potential hazards or spurious transitions on their outputs are *marked*.

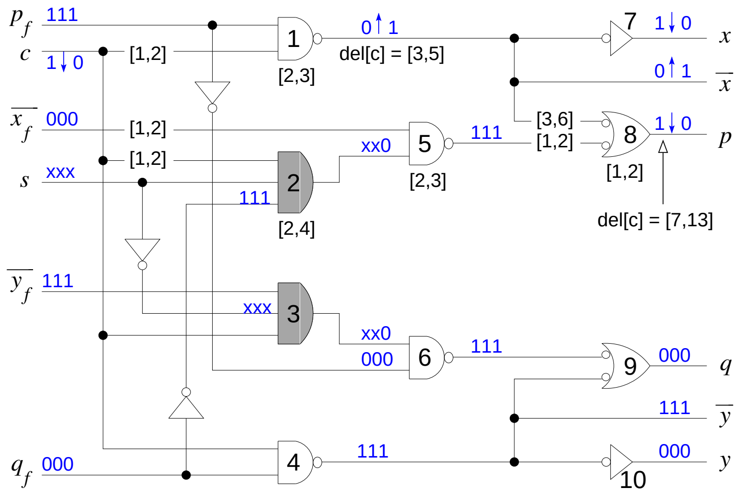
2. Next, it is assumed that a particular timing constraint is violated. Accordingly, appropriate 13-valued stimuli are applied at the inputs of the combinational circuit and the circuit re-simulated.

Gates that were not marked in step 1, but have potential hazards on their outputs in step 2 are identified as the problem gates for the timing constraint violation simulated in step 2. To see why this is so, note that gates that were marked in step 1 had potentially spurious transitions on their outputs despite all timing constraints being satisfied. 3D design guarantees that these spurious transitions will eventually get masked before reaching the outputs of the circuit. Therefore, these gates are not responsible for malfunctioning of the circuit; instead, gates that were free of spurious transitions in the first pass, but have potential hazards on their outputs in the second pass are the ones where the timing constraint violation potentially manifests itself.

For each pass of 13-valued simulation, the stimuli at the inputs of the combinational circuit are obtained as follows. When simulating the first phase of operation of the 3D machine, all feedback inputs are assumed to be stable at values determined by the current state of the machine. During the second and third phases, the transitions on the feedback outputs are copied over to the corresponding feedback inputs in order to model the effects of feedback transitions. The 13-valued waveforms at the primary inputs of the 3D circuit are obtained from the input burst that enables the state transition. Compulsory transitions, with a $+$ ($-$) sign are modeled by a $\langle 0, \uparrow, 1\rangle$ ($\langle 1, \downarrow, 0\rangle$) waveform. Directed don't cares and non-compulsory terminating edges are modeled in the same way assuming they are transitioning in the appropriate direction. The rationale behind this is that 13-valued simulation considers all possible orderings of the input transitions (see Section 2.3), so any hazard or spurious transition generated with a stable input will also be detected by 13-valued simulation that considers the input as transitioning. In addition, spurious transitions that would not have occurred with a stable input may be detected if we allow the input to transition. Therefore, modeling directed don't cares and non-compulsory terminating edges as transitioning inputs covers the worst case situation for generation of spurious transitions. Conditional signals that are sampled are modeled by $\langle X, X, 0\rangle$ or $\langle X, X, 1\rangle$ (depending on the value being sampled), when determining setup-time constraints. This represents the fact that the signal stabilizes to a binary value from an unknown initial state. When determining hold-time constraints, the sampled conditionals are modeled as $\langle 0, X, X\rangle$ or $\langle 1, X, X\rangle$, representing the fact that the signal becomes undefined from a stable binary value. Conditionals that are not sampled are modeled as $\langle X, X, X\rangle$, representing their undefined state.

As an example, consider the 13-valued annotations in the circuit in Fig. 3.4 (the delay bounds shown alongside the gates and wires will be needed later in Section 3.5). This circuit is the same as the 3D circuit in Fig. 3.2b, but with the feedback loops separated into feedback outputs and feedback inputs. Suppose we wish to simulate the state transition $S_1 \overset{c-/x-}{\longrightarrow} S_0$ (see Fig. 3.1) and identify problem gates where minimum feedback delay violations are manifested, under the assumption that $p_f = \overline{y_f} = \langle 1, 1, 1\rangle$ and $q_f = \overline{x_f} = \langle 0, 0, 0\rangle$ in state S_1 (it can be easily verified that these are the unique stable values of the feedback inputs in state S_1).

The first phase of operation of the 3D machine is simulated as follows. Since no conditionals appear in the input burst of $S_1 \overset{c-/x-}{\rightarrow} S_0$, the conditional input s is set to $\langle X, X, X\rangle$ since it can transition freely. Edge input c is set to $\langle 1, \downarrow, 0\rangle$. The result of the 13-valued simulation is shown in Fig. 3.4. Gates 2 and 3 potentially have spurious transitions on their



Gate and wire delays shown are [min, max].

Figure 3.4: 13-valued simulation and min-max timing analysis.

outputs ($\langle X, X, 0\rangle$) although no global timing constraints have been violated. Therefore, these gates are marked (shown shaded in Fig. 3.4).

Next, the second phase of operation of the machine is simulated. The $\langle 1, \downarrow, 0\rangle$ transition on feedback output p in Fig. 3.4 is copied over to p_f and similarly, the $\langle 0, \uparrow, 1\rangle$ on $\overline{x}$ is copied over to $\overline{x_f}$. The assumption that all global timing constraints are satisfied implies, in this case, that there is sufficient delay in the feedback paths. Consequently, input c would have settled to $\langle 0, 0, 0\rangle$ by the time the transitions on p and $\overline{x}$ reach p_f and $\overline{x_f}$. Therefore, c is set to $\langle 0, 0, 0\rangle$, s remains at $\langle X, X, X\rangle$ (it can still transition freely) and the circuit is re-simulated. The resulting situation is shown in Figure 3.5. Observe that there are no gates with spurious transitions or hazards on their outputs, so no additional gates are marked in this step.

Finally, to relax the assumption about feedback paths having sufficient delays, the circuit is re-simulated with a $\langle 1, \downarrow, 0\rangle$ transition on c, a $\langle 0, \uparrow, 1\rangle$ transition on $\overline{x_f}$ and a $\langle 1, \downarrow, 0\rangle$ transition on p_f. This models the fact that the transition on c could potentially interact with the transitions on the feedback inputs $\overline{x_f}$ and p_f at some internal gate if the feedback paths were fast enough. The result of the simulation is shown as 13-valued annotations in

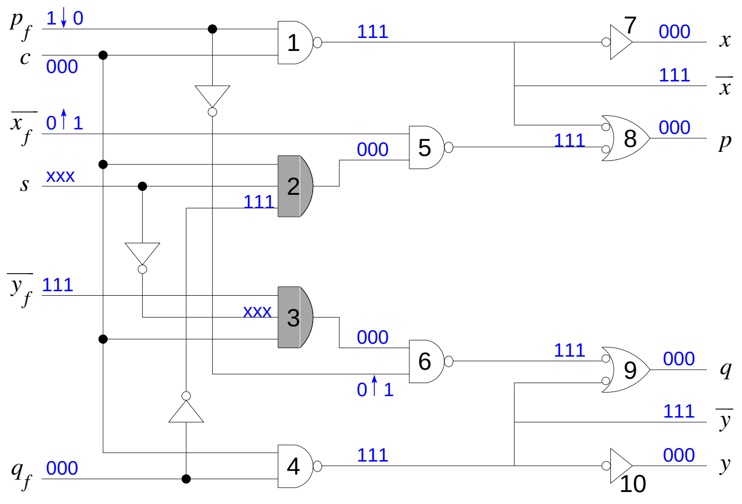
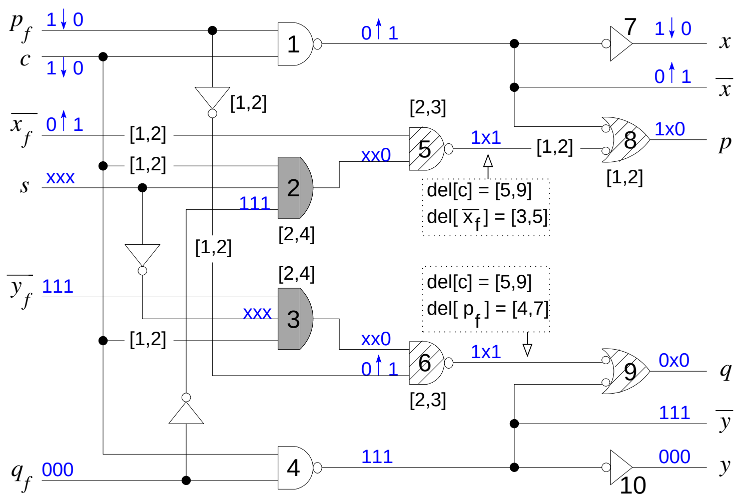


Figure 3.5: 13-valued simulation of second phase of operation.



Gate and wire delays shown are [min, max].

Figure 3.6: Identifying problem gates, and min-max timing analysis.

Fig. 3.6. Gates 5, 6, 8 and 9 were not marked in earlier passes, but exhibit potential hazards now. Therefore, these are the problem gates where insufficient feedback delays can cause spurious transitions during the state transition $S_1 \rightarrow S_0$.

3.5 Timing constraints for 3D circuits

The min-max timing simulation algorithm described in Section 2.4 is now used to compute signal propagation delay bounds from the primary and feedback inputs to the problem gates identified above. In general, a problem gate G in a 3D circuit has a spurious transition on its output because transitions propagating from two circuit inputs i and j (primary or feedback inputs) reach the gate in the wrong order. However, if $G.del[i].t$ is a lower bound on the signal propagation delay from primary input i to G, and $G.del[i].T$ is an upper bound on the same delay, the transition from j can be made to reach gate G before the transition from i by requiring that j transition at least $d = G.del[j].T - G.del[i].t$ time units before i. Since the delay bounds computed by the min-max timing simulation algorithm are conservative, the above strategy *guarantees* that the transition from j reaches G no later than the transition from i.

As an example, consider the delay annotations in the circuits in Figs. 3.4 and 3.6. Fig. 3.4 shows signal propagation delay bounds for gates 1 and 8 computed during the first phase of simulation, and Fig. 3.6 shows the bounds for gates 5 and 6 computed during the final phase of simulation.[1] The bounds in Fig. 3.6 indicate that a transition on primary input c takes a maximum of 9 time units to reach the output of gate 5, whereas a transition on $\overline{x_f}$ takes a minimum of 3 time units to reach the output of the same gate. In order to ensure that the transition on primary input c reaches gate 5 before the transition on feedback input $\overline{x_f}$ (which is required to meet the feedback delay constraint), $\overline{x_f}$ must transition at least $9 - 3 = 6$ time units after c transitions. However, $\overline{x}$ itself transitions at least 3 time units after c transitions (obtained from Fig. 3.4). Therefore, the additional feedback delay needed in the $\overline{x} \rightarrow \overline{x_f}$ path is $6 - 3 = 3$ time units. A similar analysis for gate 6 indicates that p_f must transition at least 5 time units after c transitions. However, we know from

[1] For clarity, not all delay bounds are shown in each figure.

Fig. 3.4 that p transitions at least 7 time units after c transitions. Consequently, no additional delay in the $p \rightarrow p_f$ path is needed to meet the feedback delay constraint. This shows how inherent circuit delays may obviate the need for inserting additional feedback delays. Note that while gates 5, 6, 8 and 9 are designated as problem gates in Fig. 3.6, ensuring the absence of hazards at gates 5 and 6 automatically eliminates any hazards at gates 8 and 9. Therefore, gates 8 and 9 do not give rise to additional feedback delay constraints.

The method for determining minimum feedback delays described above can be easily extended to determine setup-time and hold-time constraints in 3D circuits. Setup-time constraints are obtained by ensuring that the desired values of all sampled conditionals are available at the inputs of each problem gate before a sampling edge reach the same gate. Hold-time constraints are analyzed by ensuring that an edge that de-sensitizes a problem gate to a transition propagating from a conditional input reaches the gate before a change in the conditional signal (from its sampled value to an undefined value) propagates to the same gate. The de-sensitizing edge must propagate through the feedback path before it reaches the problem gate, so the values of hold-time constraints cannot be determined until the feedback delays are known. To resolve this dependency, the algorithm remembers the feedback paths through which the de-sensitizing edges must propagate in order to reach a problem gate. The hold-time constraint is then expressed in terms of signal propagation delays through the combinational circuit and the feedback paths involved. Once all state transitions in the extended burst-mode specification have been analyzed and the feedback delays determined, the computed delays are used to determine the values of the hold-time constraints. Figs. 3.7a and b illustrates this idea for determining setup and hold-time constraints.

In order to determine fundamental-mode constraints, the algorithm simulates each pair of consecutive state transitions, $S_0 \rightarrow S_1 \rightarrow S_2$, in the extended burst-mode specification. The basic technique is similar to that used for determining feedback delay constraints. The only difference from the previous checks is that interactions between transitioning feedback signals of the $S_0 \rightarrow S_1$ transition and transitioning primary inputs of the $S_1 \rightarrow S_2$ transition are now modeled. As with hold-times, the algorithm remembers the feedback paths through which the signal transitions from the $S_0 \rightarrow S_1$ state transition propagate, and express the fundamental-mode constraints in terms of signal propagation delays through the

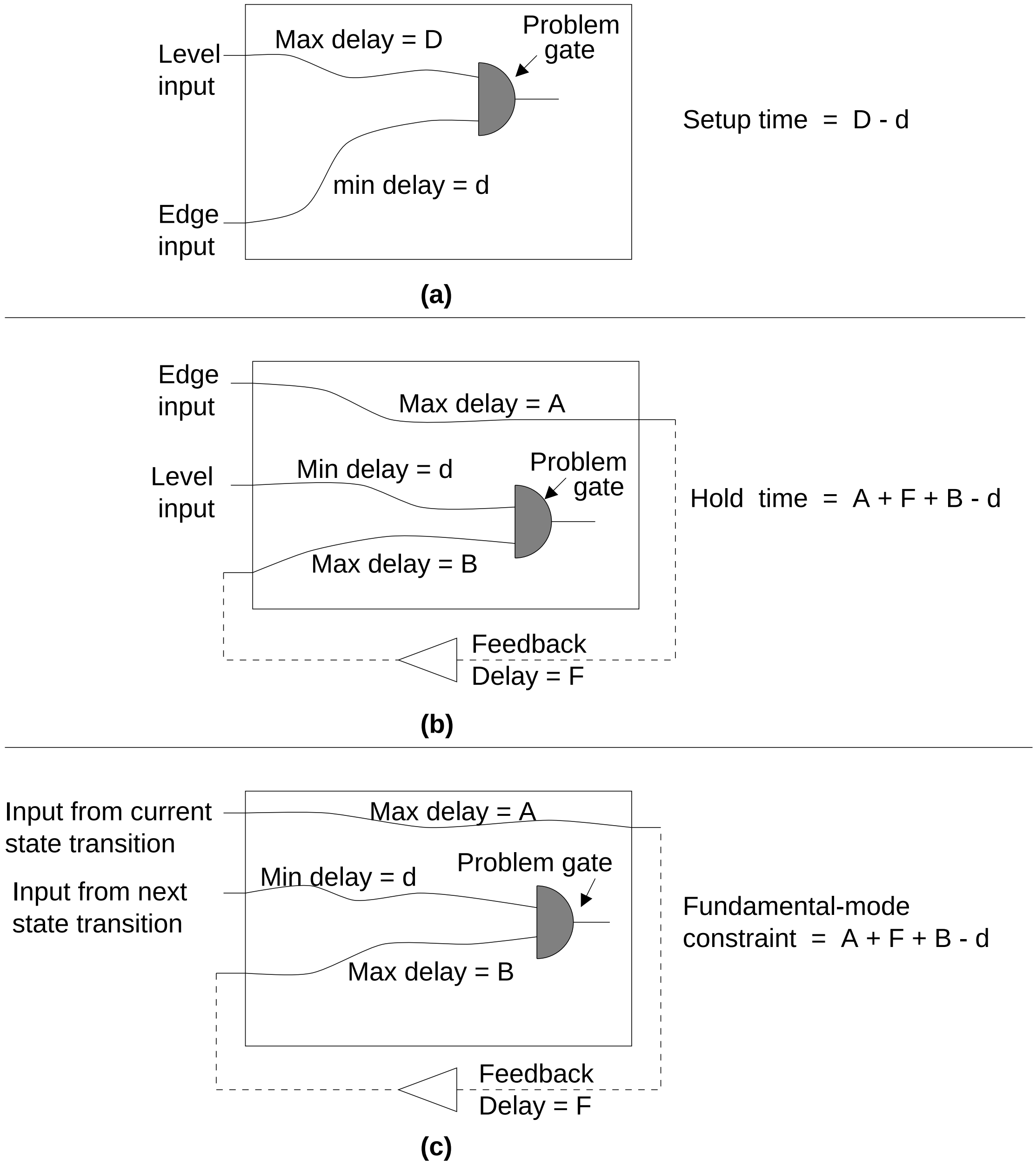


Figure 3.7: (a) Determining setup-times; (b) Determining hold-times; (c) Determining fundamental-mode constraints.

combinational circuit and the feedback paths. Once the feedback delays are determined, the values of the fundamental-mode constraints are obtained. Fig. 3.7c illustrates this idea for determining fundamental-mode constraints.

3.6 3D Timing Analysis Tool

The ideas described above have been implemented in an automatic timing analysis tool for 3D asynchronous circuits. The tool accepts two inputs: (i) an extended burst-mode specification, and (ii) a 3D implementation with delay bounds annotated on each gate. It computes safe bounds on setup and hold-time constraints, minimum feedback delays and fundamental-mode constraints for correct operation of the 3D circuit. The structure of the tool is shown in Fig. 3.8.

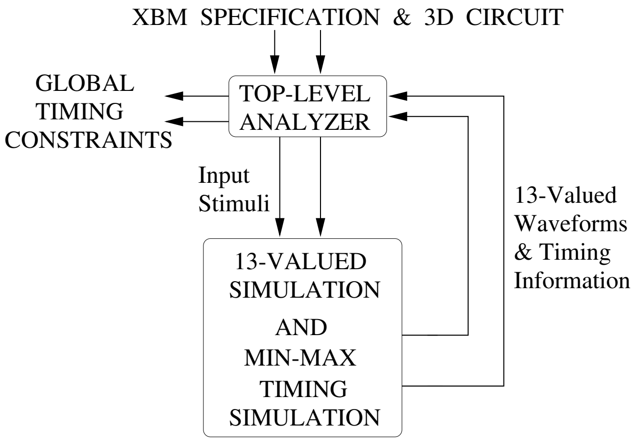


Figure 3.8: Timing analysis tool for 3D circuits.

The top-level analyzer cuts the feedback loops to obtain a combinational circuit. For each phase of each extended burst-mode state transition, it then determines the 13-valued stimuli to apply to the inputs of the combinational circuit, and invokes the 13-valued simulator. Problem gates for each type of global timing constraint violation are identified from the results of 13-valued simulation. The min-max timing simulator is then invoked, and the computed delay bounds are fed back to the top-level analyzer. Timing information and 13-valued output waveforms computed by the timing simulator are used by the top-level analyzer to determine: (a) timing constraints for correct operation of the circuit, and (b) 13-valued waveforms and timing information to copy from the feedback outputs to the corresponding feedback inputs. These are passed on to the 13-valued simulator and min-max timing simulator for the next pass of analysis. The iteration continues until each extended burst-mode state transition has been examined for setup and hold-time and feedback delay violations, and each pair of consecutive state transitions have been analyzed for fundamental-mode constraint violations.

The worst case complexity of the timing analysis technique is $O(n^2_{transitions}.C_{min-max})$, where $n_{transitions}$ represents the number of state transitions in the extended burst-mode specification and $C_{min-max}$ is the complexity of the min-max timing simulation algorithm. As described in Section 2.4.3. $C_{min-max}$ is a polynomial in the size of the circuit.

3.7 Experimental Results

The tool described above has been evaluated by applying it to a suite of extended burst-mode benchmarks synthesized according to the 3D design style. The goal of the experiments was to evaluate the speed and accuracy of the tool. While one might argue about the input parameters used, e.g. percentage variations of delays, the algorithm is not terribly sensitive to them.

The first four columns in Table 3.1 give the number of states, state transitions, primary inputs and gate count of each benchmark. The column labeled “Analysis Time” gives the time taken by the tool to exercise all extended burst-mode state transitions and determine global timing constraints for each benchmark. The CPU times shown are on a DEC 5000/240 machine, and do not include the time required to read in the extended burst-mode specification and 3D circuit description, and topologically sort the gates, which are negligible.

The benchmarks labeled “ircv”, “isend”, “trcv” and “tsend” are specifications of different components of a SCSI data transfer controller. The mode of operation of each controller is indicated by its tag. The benchmarks tagged “-bm” operate in burst-mode, those tagged

"-csm" operate in a different mode called cycle-steal-mode, while the untagged benchmarks can operate in both modes. "biu-dma2fifo" represents a Bus Interface Unit (BIU) controller for transferring data from a DMA bus to a FIFO buffer in the SCSI controller. "biu-fifo2dma" specifies the BIU controller for transferring data from the FIFO buffer in the SCSI controller to the DMA bus. The benchmarks labeled "scsi-init-send" and "scsi-targ-send" also specify components of a SCSI controller. The benchmark "pscsi" is a complete pipelined SCSI controller.

3D Benchmark	States	State Transitions	Inputs	Gates	Analysis Time(s)
ircv	16	22	8	107	1.375
ircv-bm	8	10	7	58	0.309
ircv-csm	8	10	7	64	0.339
isend	24	32	8	247	7.738
isend-bm	12	15	7	104	1.052
isend-csm	12	15	7	81	0.698
trcv	16	22	8	96	1.318
trcv-bm	8	10	7	58	0.332
trcv-csm	8	10	7	59	0.328
tsend	22	30	8	182	4.761
tsend-bm	11	14	7	65	0.466
tsend-csm	11	14	7	69	0.492
biu-dma2fifo	12	15	6	81	0.711
biu-fifo2dma	11	13	6	77	0.590
scsi-init-send	7	8	5	33	0.105
scsi-targ-send	7	8	5	41	0.155
pscsi	45	62	11	350	17.171

Table 3.1: 3D benchmark characteristics and analysis times.

The nominal delays of the gates are estimated using a Hitachi CMOS gate library [1], as described in Section 2.6. The gate delays used in the experiments are assumed to vary within $(0.9d, 1.1d)$, where d is the nominal gate delay. The reader may wonder about the usefulness of considering $\pm 10\%$ delay variations, but it doesn't matter much for the experiments. In fact, the percentage variation is an adjustable parameter of the tool that can be modified to examine the effects of other delay variations. Wire delays are assumed to be zero, since post-layout information about wire delays were not available.

3D Benchmark	Minm. feedback (FO4 del)	Setup Time (FO4 del)	Hold Time (FO4 del)	Fnd-mode constrnt. (FO4 del)
ircv	0.000	1.361	7.140	4.789
ircv-bm	0.000	0.000	4.488	1.959
ircv-csm	0.000	0.519	4.015	1.497
isend	0.000	1.028	8.189	3.996
isend-bm	0.000	0.492	7.231	3.751
isend-csm	0.000	0.924	6.226	3.589
trcv	0.000	0.995	5.594	2.715
trcv-bm	0.243	0.080	5.501	2.073
trcv-csm	0.000	0.681	3.775	1.395
tsend	0.000	2.632	7.777	3.403
tsend-bm	0.000	0.371	5.860	2.600
tsend-csm	0.000	0.757	3.984	1.102
biu-dma2fifo	0.000	0.213	7.390	2.550
biu-fifo2dma	0.298	0.592	4.753	1.357
scsi-init-send	0.000	0.213	2.564	0.044
scsi-targ-send	0.000	0.000	4.172	0.540
pscsi	0.000	0.000	0.000	3.505

Table 3.2: Results of timing analysis of 3D circuits.

The timing constraints obtained for each benchmark are reported in Table 3.2. All constraints are normalized with respect to the delay of an inverter with fanout 4 (FO4 delay). “Minm. feedback” is obtained by determining the minimum delay required in each feedback path for each extended burst-mode state transition, and then taking the maximum of these delays over all state transitions and all feedback paths. “Setup Time” is obtained by determining the setup time for each sampled conditional signal in each extended burst-mode state transition, and then taking the maximum of these values over all state transitions and all conditional inputs. “Hold Time” is similarly computed. The tool reports setup and hold-times of 0 when there are no conditional inputs (e.g., “pscsi”). To obtain fundamental-mode constraints, the minimum delay that must elapse between the last primary output transition of the present burst and the first compulsory input transition of the next burst is determined for each pair of consecutive state transitions. "Fnd-mode constraint" is then obtained by taking the maximum of these values over all pairs of consecutive state transitions.

The accuracy of the results has been manually checked for all benchmarks except "pscsi" by running the tool in an interactive debugging mode. In this mode, whenever a timing constraint is detected and is larger than was previously detected, the tool prints out the state of the entire circuit with delay annotations. This is then manually checked to see if the timing constraint reported by the tool is exact. Although this process is very painstaking, it is feasible for the benchmarks shown since the sizes of the circuits involved are not huge. The benchmark "pscsi" is, however, too large for manual inspection. The manual checks have shown that the timing constraints identified by the tool are accurate for all the 3D benchmarks, except "pscsi" which could not be manually checked. However, even for "pscsi", the tool *conservatively* estimates that no additional delays are needed in the feedback paths. Therefore, feedback delays are definitely not needed in this circuit. The fundamental-mode constraint for "pscsi" also seems quite reasonable (approximately 3.5 inverter delays).

As an interesting aside, the analysis shows that for practical 3D circuits, additional feedback delays are almost never needed – the inherent circuit delays are sufficient to ensure that essential hazards don't occur. Setup-time constraints also seem reasonable (of the order of one gate delay). However, the maximum hold-times are large since the machine often has to wait for feedback signal transitions in the third phase of operation for some internal gates to become de-sensitized to the conditionals. Note that this is an artifact of the design style, and not of the timing analysis tool. Fundamental-mode timing constraints for the 3D benchmarks also seem reasonable, since the environment will require a few gate delays' time to react to the primary output transitions.

3.8 Summary

This chapter described an efficient timing analysis tool for determining global timing constraints for correct operation of extended burst-mode circuits implemented according to the 3D design style. The tool automatically extracts information about 3D circuits similar to the information that appears in the "data sheet" provided by a component manufacturer. Although timing constraints identified by the tool represent conservative approximations to the true timing requirements in the worst case, experimental results indicate that the results are fairly accurate. The efficiency and accuracy of the tool makes it attractive for use in a design-analyze-redesign setting. This can prove very helpful in the design cycle of practical 3D circuits.

The work described in this chapter can potentially be extended to other interesting applications. For example, one could modify the timing analysis algorithm to verify the correctness of a 3D design by simulation. Another interesting application would be to determine if the gate and wire delays in a 3D circuit permit the elimination of some hardware required for ensuring hazard-free behavior. This could be used, for example, to optimize the area of a 3D circuit.

Chapter 4

Time Separation of Events: Acyclic Systems

4.1 Introduction

The behavior of asynchronous systems is often described in terms of events and their interactions. An *event* represents an action, such as a signal transition, that is assumed to be atomic — at any time instant, it has either occurred or it hasn't [95]. A central problem in the analysis of asynchronous and concurrent systems is the determination of minimum and maximum separations between times of occurrence of events. The results of such an analysis can be used for interface timing verification [15, 39, 9, 72, 92, 97], synthesis, optimization and verification of asynchronous circuits [62, 78, 91, 5, 46, 22], performance analysis and scheduling of concurrent and embedded systems [24, 46, 98, 22], optimal clock scheduling in circuits with latches [17], etc.

Unfortunately, finding exact bounds on the time separation of events is computationally intractable when component delays are bounded and the times of occurrence of events are related by min and max constraints [72]. This chapter describes a *polynomial-time* algorithm for computing *approximate* bounds on the time separation of events in such systems. For efficiency, the analysis is currently restricted to systems without choice or repeated occurrences of events. A *choice* represents a situation where the system has more than one possible behaviors; the actual behavior is determined by a choice made either by the system or by its environment. Although the restriction to choice-free systems is significant, there exist several important applications that can be modeled and analyzed as choice-free systems. The analysis of systems with repeated occurrences of events is the topic of the next chapter.

Given a system of n events, let the times of occurrence of the events be represented by n variables. A timing constraint specifies a mathematical relationship between these variables. The *feasible region* of a set of timing constraints is the set of points in n-dimensional real space, henceforth called $\Re^n$, that satisfy all the constraints. If every constraint is of the form of a linear inequality between the variables, the feasible region is a *convex* polyhedron (possibly empty or unbounded) in $\Re^n$. For systems with convex feasible regions that are non-empty, efficient convex optimization techniques, such as linear programming or the shortest-path algorithm, can be used to determine bounds on the differences between the time variables. However, if the system has min and/or max constraints, the feasible region is non-convex in general, rendering convex optimization techniques inapplicable. In this chapter, such systems are efficiently analyzed by first approximating the non-convex feasible region with a convex *over-approximation* or *envelope*, and then computing bounds on the differences between the time variables in the convex envelope. Since every point in the feasible region is contained in the envelope, maximizing the separation between a pair of variables in the envelope gives an upper bound on the maximum separation in the original feasible region. Similarly, minimizing the separation between a pair of variables in the envelope gives a lower bound on the minimum separation in the original feasible region.

To ensure that the bounds computed by the above method are not overly pessimistic, it is important to minimize the volume of the region that lies outside the original feasible region but within the convex envelope. Ideally, one would like to find the smallest convex envelope — the convex hull — of the feasible region. Unfortunately, finding the convex hull of an n-dimensional region has worst-case complexity exponential in n. Therefore, for efficiency, a new, more efficient technique for obtaining a small convex envelope of the feasible region is proposed. This technique is based on approximating each min and max constraint by a system of linear inequalities.

The proposed algorithm finds exact bounds on the time separation of events when only max or only min constraints are present. For systems with both min and max constraints, the computed bounds are conservative in the worst-case; nevertheless, experiments indicate that they are fairly accurate in practice. The running times for several large examples compare favorably with an exact algorithm [72], especially when both min and max constraints are present.

The remainder of this chapter is organized as follows. Section 4.2 surveys related work on time separations analysis of systems without repeated occurrences of events. Section 4.3 describes acyclic timing constraint graphs – a formalism for representing timing constraints in systems without choice or repeated events – and formalizes the time separation of events problem for these systems. Section 4.4 describes a strategy for approximating min and max constraints using combinations of linear inequalities. Section 4.5 presents a polynomial-time algorithm for computing bounds on the time separations of all pairs of events. Section 4.6 describes a pathological example, where the proposed algorithm fails to produce exact results. Section 4.7 presents experimental results demonstrating the accuracy and efficiency of the algorithm on several examples. Finally, Section 4.8 summarizes the contributions of this chapter.

4.2 Related Work

McMillan and Dill [72] have shown that computing exact bounds on the time separation of events is NP-complete when both min and max constraints are present. They represented a *3-SAT* formula by a system of min and max constraints, and showed that the formula is satisfiable if and only if the maximum separation between a pair of events equals a specified value. They also showed that the problem is solvable in NP-time. Burks and Sakallah [17] proved a more general version of the problem, called *min-max linear programming*, to be NP-complete using reduction from *0-1 integer programming*. The literature contains several algorithms for analyzing systems with different combinations of timing constraints, such as linear-only, max-only, max and linear, or min, max and linear. The complexities of these algorithms range from polynomial time for the linear-only case to worst case exponential time for the min, max and linear case. This section briefly reviews some representative works.

Borriello [9] modeled the timing characteristics of interface signals using linear inequalities, and proposed an exact timing verification technique based on the all-pairs shortest paths algorithm [28]. Gahlinger [39], Brzozowski, Gahlinger and Mavaddat [16] and Mavaddat and Gahlinger [69] also used linear constraints for verifying the consistency of interface timing specifications and for checking the satisfiability of interface timing requirements. In their formulation, linear constraints are represented by a directed, labeled graph, in which each event is represented by a vertex and each constraint is represented by a pair of edges. For a system with V events, exact bounds on the time separations of all pairs of events are computed in $O(V^3)$ time using the all-pairs shortest paths algorithm.

McMillan and Dill [72] proposed a polynomial-time algorithm for computing time separation bounds in systems with only max type timing constraints. In their formulation, timing constraints are represented by a directed acyclic graph, in which each event is represented by a vertex and each constraint is represented by an edge labeled with a delay interval. For a graph with V vertices and E edges, their algorithm computes exact bounds on the time separations between all pairs of events in $O(V.E)$ time. The authors also proposed a *pseudo-polynomial-time* algorithm for computing exact time separation bounds in systems with both max and linear constraints. The complexity of their algorithm for max and linear systems is $O(V^3.S)$, where S denotes the sum of the edge delay bounds. For systems with min, max and linear constraints, a heuristic branch-and-bound technique was also proposed in the same work [72]. In this technique, each min constraint is eliminated by creating new subproblems in which one argument of the min is assumed to be smaller than the others by adding appropriate linear constraints. The resulting subproblems have only max and linear constraints, and are solved using the pseudo-polynomial time algorithm for max and linear systems. The results of the individual subproblems are then combined to obtain the time separation bounds of the original system. The number of subproblems generated by this technique is exponential in the number of min constraints.

Vanbekbergen et al [91] analyzed *Signal Transition Graphs (STG)* with delay intervals associated with the places of the STG, for applications in asynchronous circuit synthesis. In their terminology, linear, max and min constraints are referred to as constraints of types 1, 2 and 3, respectively. For an STG with V transitions and constraints of types 2 and 3, they proposed a recursive algorithm for computing the maximum separation between a given pair of transitions in $O(V^2)$ time. Unfortunately, their algorithm is incorrect when both min and max constraints are present [72]. However, it gives exact bounds for max-only systems.

Burks and Sakallah defined the *min-max linear programming* (mmLP) problem [17], and showed that the problem of finding bounds on the time separation of events can be formulated as an mmLP problem. They presented two exact techniques for solving the general mmLP problem. Their first method is an optimized branch-and-bound algorithm, that is essentially an elaboration of the algorithms of McMillan and Dill [72] and Lavagno and Sangiovanni-Vincentelli [62]. Their second method casts the mmLP problem as a mixed integer linear program (MILP), which is then solved using existing MILP solvers, such as *lp_solve* [7]. Although both techniques are exact, their worst case complexities are exponential in the size of the system.

Walkup and Borriello [93, 92] proposed the ShortCircuit algorithm for analyzing systems of max and linear constraints, with applications to interface timing verification and interface logic synthesis. In their work, cyclic constraint graphs are used to represent timing constraints between events. Their algorithm initializes the maximum separation between events to a large symbolic constant and then determines the set of constraints that reduce these bounds by the maximum amount. During this process, a *dependency graph* is constructed to remember the constraints that were used to obtain the most recent bounds. A strongly connected components analysis is then performed on the dependency graph to determine the set of constraints that can be cyclically applied to reduce the bounds repeatedly. A "short-circuiting" technique is then proposed to bypass the repeated application of these constraints. For a constraint graph with V vertices and E edges, the authors *conjecture* that the time taken to compute exact bounds on the separations between all pairs of events is $O(V^2.E^2)$. It has been demonstrated that the ShortCircuit algorithm outperforms McMillan and Dill's pseudo-polynomial-time algorithm [72] for max and linear systems when *false negative cycles* are present in the constraint graph [92].

Yen et al [97] have proposed the MaxSeparation algorithm for analyzing systems of max and linear constraints. In their method, a *compulsory constraint graph* is first constructed to represent linear constraints that *must* be satisfied at all points in the feasible region. This effectively defines a convex over-approximation of the feasible region. The longest-path algorithm [28] is then applied to the compulsory constraint graph to determine lower bounds on the maximum separation between events. Using these lower bounds as the initial estimates of the maximum separations, a *slack graph* is constructed and analyzed to determine the maximum increments by which the current estimates can be increased without violating already satisfied constraints. The estimates are accordingly refined and the slack graph re-computed and re-analyzed. The entire process is repeated until none of the estimates can be increased any more. For a constraint graph with V vertices and E edges, the authors *conjecture* that their algorithm takes $O(V^2.E + V^3.\log V)$ time to compute exact bounds on the time separations between all pairs of events. Although both the ShortCircuit algorithm and the MaxSeparation algorithm appear to have polynomial time complexity on examples, the complexity of the problem with max and linear constraints is currently unknown. There is no algorithm with a proven polynomial bound, nor is there a proof of NP-completeness.

Amon and Borriello [2] proposed a technique based on *constraint logic programming (CLP)* for timing analysis of systems with min, max and linear constraints. Subsequently, Girodias, Cerny and Older [40] have also described another method based on CLP augmented with relational interval arithmetic, for analyzing networks of linear, min and max constraints with bounded delays. Although these techniques are exact, the representation of the temporal behavior of large systems using CLP languages can be cumbersome. Moreover, the worst case complexities of these techniques are exponential in the size of the system.

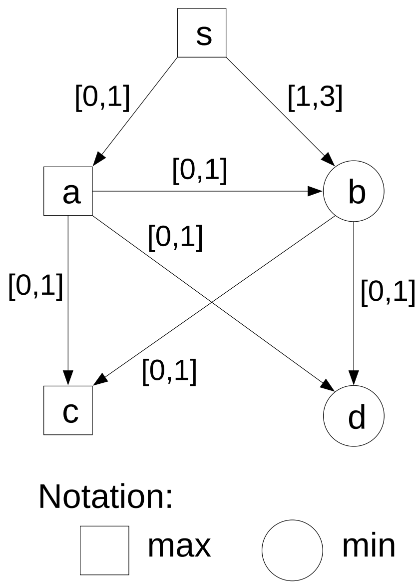
An overview of related work on time separation of events analysis for systems with repeated occurrences of events is presented in Chapter 5.

4.3 Problem Representation and Formalization

This section describes an abstract graphical formalism for representing timing constraints between events in systems without choice or repeated occurrences of event. The time separation of events problem for these systems is also formalized.

4.3.1 Acyclic Timing Constraint Graphs

Following representations used by earlier researchers [16, 9, 18, 72, 91], timing constraints between events are represented by a directed, labeled graph, $G = (V, E)$, called the *timing constraint graph*. The graphs in this chapter are assumed to be *finite* and *acyclic*. Vertices in V represent events, and edges in E represent constraints between the times of occurrence of events. In the remainder of this chapter, a vertex is referred to by the event it represents. Every event v is associated with a *min* or *max* operator, *but not both*, that determines how



Edge delays are [min, max].

Figure 4.1: Example of an acyclic timing constraint graph.

its time of occurrence depends on those of other events. Following notation used in earlier works [69, 72, 97], a min event is graphically represented as a circle and a max event is represented as a square (see Fig. 4.1). An event with no incoming edges is called a *source* event, and one with no outgoing edges is called a *sink* event. Every edge $\langle u, v \rangle$, from u to v, is labeled with an uncertain but bounded delay, $\delta_{u,v}$, that represents the delay in the propagation of the effect of event u to the component that generates event v. The edge delays are constrained to lie within fixed lower and upper bounds, $d_{u,v}$ and $D_{u,v}$, i.e., $d_{u,v} \leq \delta_{u,v} \leq D_{u,v}$.

A *path* P from u to v is 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 P. If $\langle u, v \rangle$ is an edge in the timing constraint graph, we say that event u is a *predecessor* of event v. The set of all predecessors of v is called $preds(v)$. The time of occurrence of event v is denoted τ_v and depends on the times of occurrence of its predecessors and the corresponding edge delays. Specifically, if v is associated with a min operator, then τ_v is given by $\min_{u \in preds(v)}(\tau_u + \delta_{u,v})$. For a max type event, the expression for τ_v is similar, with the min function replaced by a max function.

Previous researchers [16, 39, 9, 72, 91, 92, 97] have also considered another type of constraint, called a *linear* constraint, of the form $d_{u,v} \leq \tau_v - \tau_u \leq D_{u,v}$. In this chapter, if both τ_u and τ_v depend on other time variables through min or max constraints, additional linear constraints between τ_u and τ_v, of the form above, are not allowed. This is a limitation of the current work. Although it seems likely that the proposed algorithm can be easily modified to obtain approximate bounds in systems with min, max and linear constraints, a thorough investigation of this issue remains to be done.

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

Given two events u and v, the *maximum achievable time separation* from u to v, denoted $\Delta^*(u, v)$, is a real number with the following properties:

1. For all consistent timing assignments, $(\tau_v - \tau_u)$ is no larger than $\Delta^*(u, v)$, and

2. There exists at least one timing assignment for which $(\tau_v - \tau_u)$ is equal to $\Delta^*(u, v)$.

The *minimum achievable time separation* from u to v can be similarly defined, and is easily seen to be the same as $-\Delta^*(v, u)$.

Example 4.1 *The timing constraints represented by the acyclic timing constraint graph in Figure 4.1 are summarized below.*

$$\begin{aligned}
\tau_a &= \tau_s + \delta_{s,a} \\
\tau_b &= \min(\tau_s + \delta_{s,b},\ \tau_a + \delta_{a,b}) \\
\tau_c &= \max(\tau_a + \delta_{a,c},\ \tau_b + \delta_{b,c}) \\
\tau_d &= \min(\tau_a + \delta_{a,d},\ \tau_b + \delta_{b,d}) \\
0 &\le \delta_{s,a},\ \delta_{a,b},\ \delta_{a,c},\ \delta_{b,c},\ \delta_{a,d},\ \delta_{b,d} \le 1 \\
1 &\le \delta_{s,b} \le 3.
\end{aligned}$$

A consistent timing assignment for this system is:

$$\tau_s = 0,\ \tau_a = 1,\ \tau_b = 2,\ \tau_c = 3,\ \tau_d = 2.$$

It may be verified that the maximum achievable time separation from s to the other events are:

$$\Delta^*(s,a) = 1,\ \Delta^*(s,b) = 2,\ \Delta^*(s,c) = 3,\ \Delta^*(s,d) = 2.$$

Note that these separations are achieved by the above timing assignment.

4.3.2 The Problem

Given an acyclic timing constraint graph, $G = (V, E)$, the goal is to compute upper and lower bounds on $(\tau_v - \tau_u)$, for every pair of events u and v in V. Since a lower bound on $(\tau_v - \tau_u)$ is also an upper bound on $(\tau_u - \tau_v)$ with a change in sign, the problem can be formulated as follows:

Determine an upper bound on $(\tau_v - \tau_u)$ for every ***ordered pair*** *of events (u, v) in V.*

4.4 Approximation Strategy

This section describes a technique for approximating min and max constraints using systems of linear inequalities. The approximation strategy ensures that every point in the feasible region of the min or max constraint is contained in the feasible region of the approximate system of inequalities.

Consider a max constraint of the form:

$$\begin{aligned} \tau_f &= \max(\tau_i + \delta_{i,f}, \ldots \tau_k + \delta_{k,f}), \quad \text{where} \\ & d_{i,f} \leq \delta_{i,f} \leq D_{i,f}, \\ & \qquad \vdots \\ & d_{k,f} \leq \delta_{k,f} \leq D_{k,f}. \end{aligned} \tag{4.1}$$

Mathematically, this is equivalent to the following system of linear inequalities:

$$\begin{array}{cc} (\tau_f \geq \tau_i + d_{i,f}) & \wedge \\ \vdots & \wedge \\ (\tau_f \geq \tau_k + d_{k,f}) & \wedge \\ \multicolumn{2}{c}{\left(\begin{array}{cc} (\tau_f \leq \tau_i + D_{i,f}) & \vee \\ \vdots & \vee \\ (\tau_f \leq \tau_k + D_{k,f}) & \end{array} \right).} \end{array} \tag{4.2}$$

The disjunction of inequalities in (4.2) gives rise to the non-convex shape of the feasible region. Therefore, to obtain a convex envelope of the feasible region, the max constraint must be approximated only by conjunction of linear inequalities. However, for the approximation to be good, the disjunctive constraints in (4.2) must also not be ignored. The trick is to derive a system of inequalities from (4.2), in which only conjunction is used. This is described below.

Let s be an arbitrary event in the system. For every predecessor i of f, let $\Delta(i,s)$ and $\Delta(s,i)$ be *any* values that satisfy the following properties:

$$\tau_s \leq \tau_i + \Delta(i,s), \tag{4.3}$$

$$\tau_i \leq \tau_s + \Delta(s,i). \tag{4.4}$$

The values of $\Delta(s,i)$ and $\Delta(i,s)$ can now be used in conjunction with inequalities (4.2) to derive linear relations between the time of occurrence of f and that of the arbitrary event s. Specifically,

$$\begin{array}{rcll} \tau_f & \geq & \tau_i + d_{i,f} & \text{from (4.2), and} \\ \tau_i & \geq & \tau_s - \Delta(i,s) & \text{from (4.3) imply} \\ \tau_f & \geq & \tau_s - \Delta(i,s) + d_{i,f}. & \end{array}$$

Similarly,

$$\begin{array}{rcll} \tau_f & \leq & \tau_i + D_{i,f} & \text{from (4.2), and} \\ \tau_i & \leq & \tau_s + \Delta(s,i) & \text{from (4.4) imply} \\ \tau_f & \leq & \tau_s + \Delta(s,i) + D_{i,f}. & \end{array}$$

Therefore, the inequalities in (4.2) imply the following for every event s:

$$\begin{array}{c} (\tau_f \geq \tau_s - \Delta(i,s) + d_{i,f}) \quad \wedge \\ \vdots \qquad\qquad \wedge \\ (\tau_f \geq \tau_s - \Delta(k,s) + d_{k,f}) \quad \wedge \\ \left(\begin{array}{ll} (\tau_f \leq \tau_s + \Delta(s,i) + D_{i,f}) & \vee \\ \quad\vdots & \vee \\ (\tau_f \leq \tau_s + \Delta(s,k) + D_{k,f}), & \end{array} \right) \end{array} \tag{4.5}$$

The conjunction of inequalities in (4.5) simplifies to

$$\tau_f \geq \tau_s + \max_{l \in preds(f)} (-\Delta(l,s) + d_{l,f}), \tag{4.6}$$

and the disjunction of inequalities is equivalent to

$$\tau_f \leq \tau_s + \max_{l \in preds(f)} (\Delta(s,l) + D_{l,f}). \tag{4.7}$$

Since choosing s to be the same as f in the above inequalities is uninteresting, it is henceforth assumed that s is not equal to f. Therefore, inequalities (4.6) and (4.7) can be restated as follows:

$$\forall s \neq f,$$
$$- \max_{l \,\in\, preds(f)} (\Delta(s,l) + D_{l,f}) \;\leq\; \tau_s - \tau_f \;\leq\; \min_{l \,\in\, preds(f)} (\Delta(l,s) - d_{l,f}). \quad (4.8)$$

The system of inequalities (4.8) is now used as an approximation of the max constraint in (4.1), and inequalities (4.2) are discarded. Note that there is no disjunction of inequalities in (4.8), so its feasible region is convex.

Inequalities (4.8) lead to the following important observation:

Observation 4.1 *Given a max-type event f, the value of $(\tau_s - \tau_f)$ is bounded above by $\min_{l \,\in\, preds(f)}(\Delta(l,s) - d_{l,f})$ for all events s other than f. Similarly, the value of $(\tau_f - \tau_s)$ is bounded above by $\max_{\,l \,\in\, preds(f)}\, (\Delta(s,l) + D_{l,f})$.*

This implies that if $\min_{l \,\in\, preds(f)}(\Delta(l,s) - d_{l,f})$ is less than the current estimate of the maximum value of $\tau_s - \tau_f$, then it can be used to refine the estimate. Similarly, $\max_{\,l \,\in\, preds(f)}\, (\Delta(s,l) + D_{l,f})$ can be used to refine the current estimate of the maximum value of $\tau_f - \tau_s$.

Now, consider a min constraint of the form

$$\begin{aligned} \tau_f \;&=\; \min(\tau_i + \delta_{i,f}, \;\ldots\; \tau_k + \delta_{k,f}), \quad \text{where} \\ & d_{i,f} \;\leq\; \delta_{i,f} \;\leq\; D_{i,f} \\ & \qquad\quad \vdots \\ & d_{k,f} \;\leq\; \delta_{k,f} \;\leq\; D_{k,f}. \end{aligned} \quad (4.9)$$

This is equivalent to the following system of linear inequalities:

$$\begin{array}{cc} (\tau_f \;\leq\; \tau_i + D_{i,f}) & \wedge \\ \vdots & \wedge \\ (\tau_f \;\leq\; \tau_k + D_{k,f}) & \wedge \end{array}$$

$$\left(\begin{array}{ll} (\tau_f \; \geq \; \tau_i \; + \; d_{i,f}) & \vee \\ \qquad \vdots & \vee \\ (\tau_f \; \geq \; \tau_k \; + \; d_{k,f}) & \end{array} \right). \tag{4.10}$$

Reasoning as in the case of the max constraint, the following inequalities can be derived for an arbitrary event s:

$$\begin{array}{c} \begin{array}{ll} (\tau_f \; \leq \; \tau_s \; + \; \Delta(s,i) \; + \; D_{i,f}) & \wedge \\ \qquad\qquad \vdots & \wedge \\ (\tau_f \; \leq \; \tau_s \; + \; \Delta(s,k) \; + \; D_{k,f}) & \wedge \end{array} \\ \left(\begin{array}{ll} (\tau_f \; \geq \; \tau_s \; - \; \Delta(i,s) \; + \; d_{i,f}) & \vee \\ \quad \vdots & \vee \\ (\tau_f \; \geq \; \tau_s \; - \; \Delta(k,s) \; + \; d_{k,f}), & \end{array} \right). \end{array} \tag{4.11}$$

Therefore, we have the following:

$$\begin{array}{l} \forall s \neq f, \\ \quad - \min_{l \,\in\, preds(f)} (\Delta(s,l) + D_{l,f}) \;\leq\; \tau_s - \tau_f \;\leq\; \max_{l \,\in\, preds(f)} (\Delta(l,s) - d_{l,f}). \end{array} \tag{4.12}$$

The system of inequalities (4.12) is now used as an approximation of the min constraint in (4.9), and inequalities (4.10) are discarded. Since there is no disjunction of inequalities in (4.12), its feasible region forms a convex envelope of the original feasible region.

Inequalities (4.12) lead to the following important observation:

Observation 4.2 *Given a min-type event f, the value of $(\tau_f \; - \; \tau_s)$ is bounded above by $\min_{l \,\in\, preds(f)}(\Delta(s,l) + D_{l,f})$ for all events s other than f. Similarly, the value of $(\tau_s - \tau_f)$ is bounded above by $\max_{\; l \,\in\, preds(f)} \; (\Delta(l,s) \; - \; d_{l,f})$.*

This implies that if $\min_{l \,\in\, preds(f)}(\Delta(s,l) \;+\; D_{l,f})$ is less than the current estimate of the maximum value of $\tau_f - \tau_s$, then it can be used to refine the estimate. Similarly, $\max_{\; l \,\in\, preds(f)} \; (\Delta(l,s) \; - \; d_{l,f})$ can be used to refine the current estimate of the maximum value of $\tau_s - \tau_f$.

Example 4.2 *Consider the following system of of timing constraints:*

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

Figure 4.2(a) shows the feasible region of this system. The volume contained within the

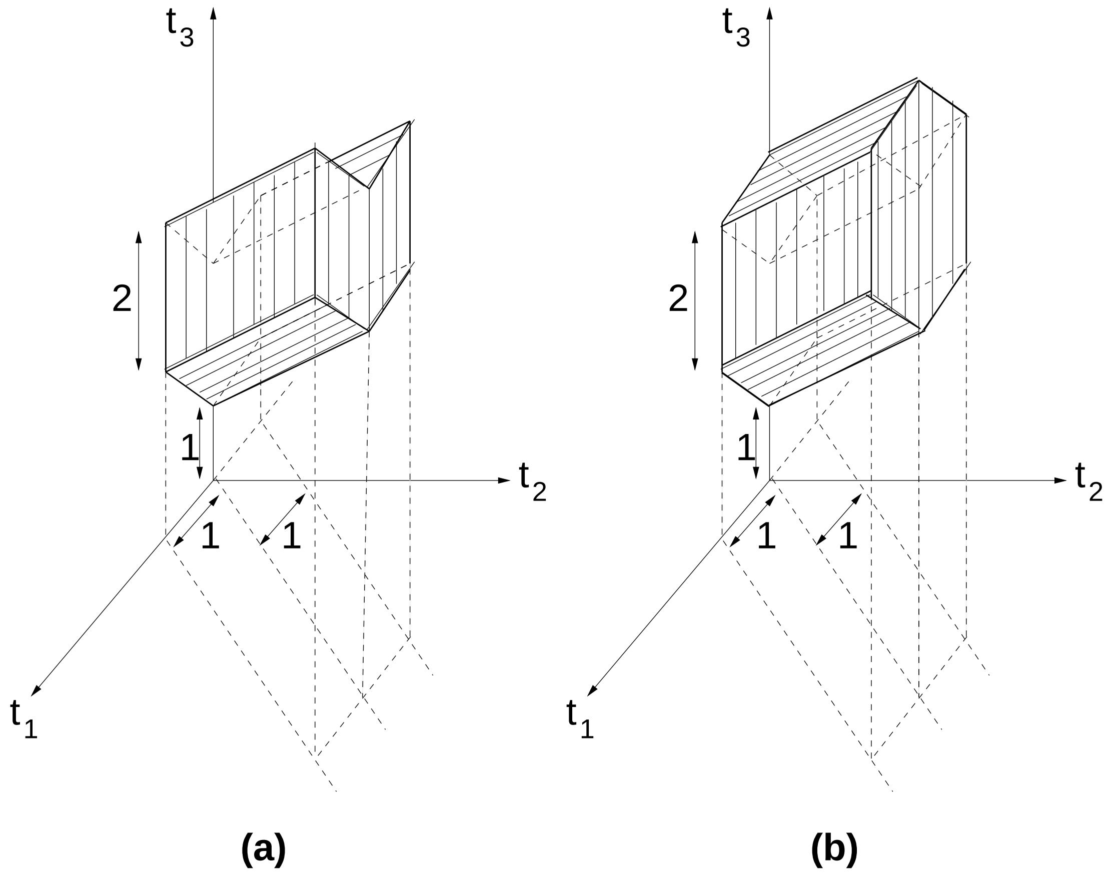


Figure 4.2: Illustrating the inclusion of the original feasible space in the approximate convex space.

"V"-shaped region and extending to infinity in both directions along the corner of the V represents the feasible region. The approximation strategy described above gives the

following linear inequalities for this system:

$$\tau_3 \geq \tau_1 + 1; \quad \tau_3 \leq \tau_1 + 4; \quad \tau_3 \geq \tau_2 + 1; \quad \tau_3 \leq \tau_2 + 4; \quad -1 \leq \tau_2 - \tau_1 \leq 1.$$

The feasible region of this system of inequalities is shown in Figure 4.2(b). Clearly, this forms a convex envelope of the original non-convex feasible region.

4.5 A Polynomial-Time Algorithm

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

In the following discussion, it is assumed that the events in the timing constraint graph are topologically sorted and every event has a topological index greater than those of its predecessors. If there are m source events, the source events are assumed to have indices 0 through $m - 1$. An event with topological index i is henceforth referred to as event i or simply i when it is clear from the context whether the index or the event is being referred to. Let us now define *layer* i of the Δ matrix as the set of all $\Delta(j, i)$ and $\Delta(i, j)$ entries, with

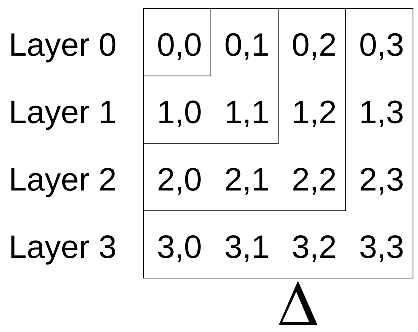


Figure 4.3: Layer wise construction of the Δ matrix.

j lying in the range 0 through i. The Δ matrix is viewed as made up of n layers, numbered 0 through $n-1$, as shown in Fig. 4.3 for a 4×4 matrix. The proposed algorithm, called AcyclicApproxSep, computes values of the entries in the Δ matrix in order of increasing layers. It starts with layer 0, i.e., $\Delta(0,0)$, which has the value 0 trivially. It then computes the entries in layer 1, followed by the entries in layer 2, and so on, until all n layers have been computed.

Fig. 4.4 shows the steps in the timing analysis algorithm[1]. During initialization, the maximum separations between all pairs of source events are set to the given bounds. The separations between all other pairs of distinct events are initialized to $+\infty$. Since the value of $\Delta(i,i)$ is 0 by definition for all i, the $\Delta(i,i)$ values are set to 0. For a system with m source events, the entries in layers 0 through $m-1$ are simply the given bounds on the separations between source events. The algorithm, therefore, computes the entries of the matrix from layer m onwards. For each layer i, the algorithm first invokes function ComputeInitialEstimates. This computes initial estimates of the entries in layer i using Observations 4.1 and 4.2 of Section 4.4. The type of event i is used to determine the expressions for the initial estimates. Referring to Fig. 4.4, note that since the value of j in steps (i) through (iv) of function ComputeInitialEstimates is less than i, and since all predecessors of event i have topological indices less than i, the Δ matrix entries used in the expressions for the initial estimates belong to layers numbered less than i. Since the entries in the matrix are computed in order of increasing layers, these entries would have already been computed by the time layer i is processed. After the initial estimates are computed, they are tightened in steps Steps (2a) through (2d) of Fig. 4.4. Once again, Observations 4.1 and 4.2 are used, but this time the type of event j is used to determine the expressions for the bounds. Note that tightening an entry in layer i requires knowledge of the values of other entries in the same layer. However, by considering each j in increasing order from 0 through $i-1$, it is ensured that the entries are estimated as well as tightened before they are required.

Let p denote the maximum number of predecessors of an event in the timing constraint graph. The complexity of the AcyclicApproxSep algorithm is given by $O(n^2.p)$. Since p

[1] This is an improved version of the algorithm reported in [19].

```
AcyclicApproxSep (G : acyclic timing constraint graph;  Δ : matrix)
 /* Let m = number of source events.                           */
/* Assume that the source events have indices 0 through m − 1 */
/* and other events have indices m through n − 1.             */

Initialize(Δ);
for each i in m to (n − 1)      /* Layer i */
  1. ComputeInitialEstimates(i, Δ);
  2. for each j in 0 to i − 1      /* Tighten estimates in layer i */
        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}));


Initialize (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});
```

Figure 4.4: Time separation of events algorithm.

is usually bounded above by a small constant, the complexity is $O(n^2)$ for all practical purposes.

The following theorems characterize the bounds computed by the algorithm. Let $\Delta^*(i,j)$ be the maximum achievable time separation from event i to event j.

Theorem 4.1 *For every ordered pair of events (i,j), $\Delta^*(i,j)$ is bounded above by $\Delta(i,j)$.*

Proof: The initialization routine guarantees that $\Delta^*(i,j)$ is bounded above by $\Delta(i,j)$, before algorithm AcyclicApproxSep starts executing.

Let us now step through the execution of the algorithm as it computes the entries in the Δ matrix. The steps are assumed to be numbered in the sequence in which they are executed, starting from 1. For notational convenience, the last step of initialization is referred to as step 0. The theorem is proved by showing that at the end of every step, $\Delta^*(i,j)$ is bounded above by $\Delta(i,j)$, for every ordered pair (i,j). The proof is by induction on the steps.

Basis: The claim is true at the end of step 0.

Hypothesis: Let the claim be true at the end of step k, where k is a non-negative integer.

Induction: Consider step $k+1$. At most one entry of the Δ matrix can be modified in this step. Let this be $\Delta(i,j)$. The following two scenarios are possible:

- If $\Delta(i,j)$ is not modified in this step, the hypothesis implies that the claim is true at the end of step $k+1$.

- If $\Delta(i,j)$ is modified in this step, the new value of $\Delta(i,j)$ is given by the expression on the right hand side of one of the assignments labeled (2a) – (2d) and (i) – (iv) in Fig. 4.4. By the hypothesis, the Δ matrix entries used in these expressions are upper bounds of the corresponding maximum achievable time separations. Therefore, by Observations 4.1 and 4.2 of Section 4.4, the newly computed value of $\Delta(i,j)$ is an upper bound of $\Delta^*(i,j)$. Therefore, the claim is true at the end of step $k+1$.

■

Theorem 4.2 *If all events are of the same type (min or max) and the timing constraint graph has a single source event, algorithm* AcyclicApproxSep *is exact. Specifically, $\Delta(i,j)$ is equal to $\Delta^*(i,j)$ for every ordered pair of events (i,j).*

Proof: Since there is a single source event, the iteration index of the outer loop of algorithm AcyclicApproxSep ranges from 1 to $n-1$. Since iteration i modifies only entries in layer i, the theorem is proved by proving the following claim.

Claim: Every entry in layer i is equal to the corresponding maximum achievable time separation at the end of iteration i.

The proof is by complete induction on the iteration index. For notational convenience, the last step of initialization is referred to as iteration 0.

Basis: After initialization, $\Delta(0,0)$ is equal to 0, which is also the value of $\Delta^*(0,0)$.

Hypothesis: Let i be an iteration index in 1 through $n-1$. For all j in 0 through $i-1$, let the claim be true at the end of iteration j.

Induction: The inductive step is first proved for max-only systems. The proof for min-only systems follows from the duality of min and max functions.

- Max-only systems: The proof is presented in three parts. In part **(I)**, $\Delta(j,i)$ is shown equal to $\Delta^*(j,i)$ for all j in 0 through $i-1$. In part **(II)**, $\Delta(i,0)$ is shown equal to $\Delta^*(i,0)$. Part **(III)** shows that $\Delta(i,j)$ is equal to $\Delta^*(i,j)$ for all j in 1 through $i-1$.

  (I): Consider $\Delta(j,i)$ where j is at most $i-1$. From step (ii) of function ComputeInitialEstimates, $\Delta(j,i)$ is *at most* $\max_{k\,\in\,preds(i)}(\Delta(j,k)+D_{k,i})$. Let event r be the predecessor of event i (or one of the predecessors, in case of a tie) with the maximum value of $\Delta(j,r)+D_{r,i}$. Since both indices r and j are less than index i, the entry $\Delta(j,r)$ belongs to a layer numbered less than i. Therefore, by hypothesis, $\Delta(j,r)$ is equal to $\Delta^*(j,r)$, and there exists at least one consistent timing assignment such that

  $$\tau_r - \tau_j \;=\; \Delta(j,r) \;=\; \Delta^*(j,r). \tag{4.13}$$

  Since events in the acyclic graph are topologically indexed and index i is greater than both indices r and j, there does not exist any path from event i to either event r or event j. This implies that the delay of the edge from r to i cannot affect the time separation between events r and j. Therefore, if we choose the delay of edge $\langle r,i\rangle$ to be $D_{r,i}$, there exists a consistent timing assignment TA

that satisfies (4.13) and in which

$$\tau_i \quad \geq \quad \tau_r + D_{r,i} \tag{4.14}$$

It follows from (4.13) and (4.14) that

$$\tau_i - \tau_j \quad \geq \quad \Delta(j,r) + D_{r,i} \tag{4.15}$$

under timing assignment TA. Since $\Delta^*(j,i)$ is at least as large as the left hand side (LHS) of (4.15) and $\Delta(j,i)$ is at most the right hand side (RHS), $\Delta^*(j,i)$ is bounded below by $\Delta(j,i)$. By Theorem 4.1, $\Delta^*(j,i)$ is also bounded above by $\Delta(j,i)$; therefore, $\Delta^*(j,i)$ is equal to $\Delta(j,i)$.

(II): Consider the entry $\Delta(i,0)$. From step (i) of ComputeInitialEstimates, the initial estimate of $\Delta(i,0)$ is $\min_{k \,\in\, preds(i)}(\Delta(k,0) \;-\; d_{k,i})$. Since event 0 has no predecessors, this estimate is not tightened subsequently; therefore, this is also the final value of $\Delta(i,0)$. Since k is a predecessor of i, the entry $\Delta(k,0)$ belongs to a layer numbered less than i. Therefore, by hypothesis, $\Delta(k,0)$ is equal to $\Delta^*(k,0)$.

Let TA be the timing assignment corresponding to τ_0 set to 0, and the delay of every edge set to the lower bound of the corresponding delay interval. Under this timing assignment, the separations from event 0 to all other events are minimized. Stated differently, the separations from all other events to event 0 are maximized. Therefore, for every predecessor k of i, we have

$$\begin{aligned} \tau_0 - \tau_k \quad &= \quad \Delta^*(k,0) \\ &= \quad \Delta(k,0) \qquad \text{(by hypothesis).} \end{aligned} \tag{4.16}$$

Since all edge delays are set to their lower bounds,

$$\begin{aligned} \tau_i \quad &= \quad \max_{k \,\in\, preds(i)} (\tau_k \;+\; d_{k,i}) \qquad \text{(by definition)} \\ &= \quad \max_{k \,\in\, preds(i)} (\tau_0 \;-\; \Delta(k,0) \;+\; d_{k,i}) \qquad \text{from (4.16)} \end{aligned}$$

$$\begin{aligned} &= \tau_0 - \min_{k \,\in\, preds(i)} (\Delta(k,0) \;-\; d_{k,i}) \\ &= \tau_0 - \Delta(i,0). \end{aligned} \tag{4.17}$$

Therefore, $(\tau_0 \;- \tau_i)$ is equal to $\Delta(i,0)$ under timing assignment TA. Hence, $\Delta^*(i,0)$ is bounded below by $\Delta(i,0)$. By Theorem 4.1, $\Delta^*(i,0)$ is also bounded above $\Delta(i,0)$; therefore, $\Delta(i,0)$ is equal to $\Delta^*(i,0)$.

(III): Finally, consider $\Delta(i,j)$ for all j lying between 0 and $i-1$. The proof that $\Delta(i,j)$ is equal to $\Delta^*(i,j)$ is by complete induction on j.

Basis: $\Delta(i,0)$ is equal to $\Delta^*(i,0)$, as shown above.

Hypothesis: Let j be an integer in 1 through $i-1$. For all s in 0 through $j-1$, let $\Delta(i,s)$ be equal to $\Delta^*(i,s)$.

Induction: From Step (2a) of algorithm AcyclicApproxSep, $\Delta(i,j)$ is *at most* $\max_{k \,\in\, preds(j)}(\Delta(i,k) \;+\; D_{k,j})$. Let event r be the predecessor of event j (or one of the predecessors, in case of a tie) with the maximum value of $\Delta(i,r) \;+\; D_{r,j}$. Since index r is less than index j, by hypothesis, $\Delta(i,r)$ is equal to $\Delta^*(i,r)$, and there exists at least one consistent timing assignment such that

$$\tau_r - \tau_i \;\;=\;\; \Delta^*(i,r) \;\;=\;\; \Delta(i,r). \tag{4.18}$$

Two different cases are now considered.

(A) Suppose there exists a consistent timing assignment TA that satisfies (4.18) and also the following inequality:

$$\tau_j \;\;\geq\;\; \tau_r + D_{r,j}. \tag{4.19}$$

Combining (4.18) and (4.18) and rearranging, we find that

$$\tau_j - \tau_i \;\;\geq\;\; \Delta(i,r) + D_{r,j} \tag{4.20}$$

under timing assignment TA. Since $\Delta^*(i,j)$ is at least as large as the LHS of (4.20) and $\Delta(i,j)$ is at most the RHS, $\Delta^*(i,j)$ is bounded below by $\Delta(i,j)$. By Theorem 4.1, $\Delta^*(i,j)$ is also bounded above by $\Delta(i,j)$. Therefore, $\Delta^*(i,j)$ is equal to $\Delta(i,j)$.

(B) Suppose *all* consistent timing assignments that satisfy (4.18) have τ_j strictly less than $\tau_r + D_{r,j}$. This implies that the delay of edge $\langle r,j\rangle$ affects the time separation from event i to event r. Now, it is easy to see that if there exist no paths from event j to either event r or event i, the delay of edge $\langle r,j\rangle$ cannot affect the time separation from event i to event r. Since there exist no paths from event j to event r (the graph is acyclic and r is a predecessor of j), this implies that there must be at least one path from event j to event i (see Fig. 4.5). Since r is a predecessor of j, the existence of paths from j

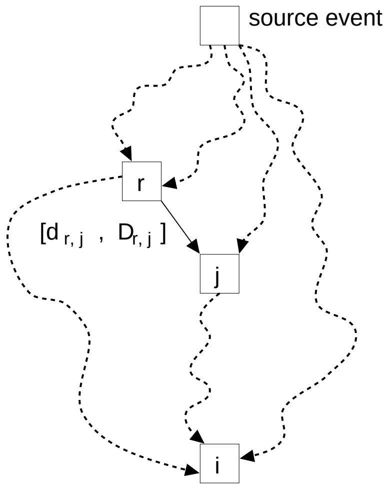


Figure 4.5: Illustrating paths in the timing constraint graph.

to i implies the existence of paths from r to i. Let the delay of every edge along every path from r to i be set to the lower bound of the corresponding delay interval. Since maximizing the separation from event i to event r is equivalent to minimizing the separation from event r to event i, there exists at least one consistent timing assignment TA that satisfies (4.18) and corresponds to the above choice of minimum edge delays along all paths from r to i. The delays of all other edges in the graph are assumed to be set to values within their respective intervals such that the time of occurrence of every event is as specified by TA. Note that since TA satisfies (4.18), by our premise, τ_j is strictly less than $\tau_r + D_{r,j}$. Therefore, the delay of edge $\langle r, j \rangle$ is less than $D_{r,j}$.

Now, consider the system that results from changing the delay of edge $\langle r, j \rangle$ to $D_{r,j}$ and keeping all other edge delays unchanged. Let TA' denote the corresponding timing assignment. For notational convenience, the time of occurrence of event v under TA' is denoted τ'_v. It is easy to see that the only events whose times of occurrence under TA' potentially differ from those under TA are event j and those to which a path from event j exists. In addition, if for any event v, the value of τ'_v is different from the value of τ_v under TA, then τ'_v must be equal to τ'_j plus the sum of edge delays along some path from j to v. Since the delay of edge $\langle r, j \rangle$ is $D_{r,j}$, the value of τ'_j is at least as large as $\tau'_r + D_{r,j}$. Therefore, by our premise, the timing assignment TA' does not satisfy (4.18). This implies that the value of $\tau_r - \tau_i$ under TA is different from the value of $\tau'_r - \tau'_i$. However, τ'_r is the same as τ_r because the delay of edge $\langle r, j \rangle$ does not affect the time of occurrence of r. Therefore, the value of τ'_i must be different from τ_i. It follows that there exists a path P from j to i such that

$$\tau'_i \quad = \quad \tau'_j + \sum_{\langle x,y \rangle \text{ along } P} d_{x,y}, \tag{4.21}$$

since all edge delays along all paths from j to i were set to their lower bounds when the edge delays along paths from r to i were assigned their minimum values. This implies that $\Delta^*(i, j)$ is bounded below by $-\sum_{\langle x,y \rangle \text{ along } P} d_{x,y}$.

Now, let w be the predecessor of i along P. From step (i) of function ComputeInitialEstimates (see Fig. 4.4), we have

$$\Delta(i, j) \quad \leq \quad \Delta(w, j) - d_{w,i} \tag{4.22}$$

If v is the predecessor of w along P, it again follows from step (i) that

$$\Delta(w, j) \;\; \leq \;\; \Delta(v, j) - d_{v,w} \tag{4.23}$$

Continuing this process and adding the resulting inequalities, we obtain:

$$\begin{aligned} \Delta(i, j) \;\; &\leq \;\; \Delta(j, j) - \sum_{\langle x,y \rangle \text{ along } P} d_{x,y} \\ &\leq \;\; - \sum_{\langle x,y \rangle \text{ along } P} d_{x,y} \end{aligned} \tag{4.24}$$

But we have shown above that the RHS of (4.24) is a lower bound of $\Delta^*(i, j)$. Therefore, $\Delta^*(i, j)$ is bounded below by $\Delta(i, j)$. By Theorem 4.1, $\Delta^*(i, j)$ is also bounded above by $\Delta(i, j)$. Therefore, $\Delta^*(i, j)$ is equal to $\Delta(i, j)$.

This completes the proof for max-only systems.

- Min-only systems: The proof for min-only systems follows from the duality of min and max functions and the proof for max-only systems. Each min constraint of the form:

$$\tau_i \;\; = \;\; \min_{k \,\in\, preds(i)} (\tau_k \; + \; \delta_{k,i}), \qquad d_{k,i} \;\; \leq \;\; \delta_{k,i} \;\; \leq \;\; D_{k,i}$$

can be rewritten as:

$$-\tau_i \;\; = \;\; \max_{k \,\in\, preds(i)} (-\tau_k \; - \; \delta_{k,i}), \qquad -D_{k,i} \;\; \leq \;\; -\delta_{k,i} \;\; \leq \;\; -d_{k,i}. \tag{4.25}$$

Using the notations τ_i', $\delta_{i,j}'$, $d_{i,j}'$ and $D_{i,j}'$ for $-\tau_i$, $-\delta_{i,j}$, $-D_{i,j}$ and $-d_{i,j}$ respectively, constraint (4.25) can be expressed as:

$$\tau_i' \;\; = \;\; \max_{k \,\in\, preds(i)} (\tau_k' \; + \; \delta_{k,i}') \qquad d_{k,i}' \;\; \leq \;\; \delta_{k,i}' \;\; \leq \;\; D_{k,i}'.$$

Therefore, a timing constraint graph with only min events can be transformed into a graph with only max events by the above transformation. Let the derived system of max constraints be called the *dual* of the original system of min constraints. Let $\Delta'(i,j)$ represent an upper bound on $(\tau'_j - \tau'_i)$ in the dual system. Note that since $(\tau'_j - \tau'_i)$ is the same as $(\tau_i - \tau_j)$, $\Delta'(i,j)$ in the dual system is equal to $\Delta(j,i)$ in the original system.

Now, consider the application of the algorithm to the dual max-only system. In step (i) of function ComputeInitialEstimates, the assignment

$$\Delta'(i,j) \;=\; \min_{k \,\in\, preds(i)} (\Delta'(k,j) \;-\; d'_{k,i})$$

is equivalent to

$$\Delta(j,i) \;=\; \min_{k \,\in\, preds(i)} (\Delta(j,k) \;+\; D_{k,i}). \tag{4.26}$$

But, this is precisely the assignment in step (iv) when the algorithm is applied to the original min-only system. Similarly, step (ii) applied to the dual max-only system is equivalent to step (iii) applied to the original min-only system. Continuing this reasoning, it is easy to see that steps (2a) and (2b) of the algorithm applied to the dual system are equivalent to steps (2d) and (2c), respectively, applied to the original system. Therefore, there exists a one-to-one correspondence between the steps of algorithm AcyclicApproxSep and function ComputeInitialEstimates when analyzing a min-only system and its dual max-only system. Since the algorithm was shown to be exact for max-only systems, and since the proof was not sensitive to the order of execution of steps (i) and (ii) in ComputeInitialEstimates, or steps (2a) and (2b) in algorithm AcyclicApproxSep, the one-to-one correspondence implies that the algorithm computes exact bounds for min-only systems.

■

Example 4.3 *Let us apply algorithm* AcyclicApproxSep *to the timing constraint graph shown in Fig. 4.1. The Δ matrix at the end of the computation is shown in Table 4.1. It may be verified that all the bounds are exact.*

	s	a	b	c	d
s	0	1	2	3	2
a	0	0	1	2	1
b	0	0	0	1	1
c	0	0	0	0	1
d	0	0	1	2	0

Table 4.1: Δ matrix for example.

It may be useful to compare the results of the above approximation with an obvious alternative. It was noted in Section 4.4 that each min and max constraint can be represented by the conjunction and disjunction of linear inequalities. As a simple approximation, one may choose to disregard the disjunctions, since taking these into account turns the feasible region non-convex. However, this can lead to problems, as can be seen by applying this simplification to the system represented in Fig 4.1. The system of linear inequalities representing this system is:

$$
\begin{array}{llllllllll}
(\tau_a & \geq & \tau_s + 0) & \wedge & & & & & & \\
(\tau_a & \leq & \tau_s + 1) & \wedge & & & & & & \\
(\tau_b & \leq & \tau_s + 3) & \wedge & & & & & & \\
(\tau_b & \leq & \tau_a + 1) & \wedge & & & & & & \\
((\tau_b & \geq & \tau_s + 1) & \vee & (\tau_b & \geq & \tau_a + 0)) & \wedge \\
(\tau_c & \geq & \tau_a + 0) & \wedge & & & & & & \\
(\tau_c & \geq & \tau_b + 0) & \wedge & & & & & & \\
((\tau_c & \leq & \tau_a + 1) & \vee & (\tau_c & \geq & \tau_b + 1)) & \wedge \\
(\tau_d & \leq & \tau_a + 1) & \wedge & & & & & & \\
(\tau_d & \leq & \tau_b + 1) & \wedge & & & & & & \\
((\tau_d & \geq & \tau_a + 0) & \vee & (\tau_d & \geq & \tau_b + 0)). & & & & &
\end{array}
$$

Disregarding the disjunctive constraints yields the following conjunction of inequalities:

$$\begin{array}{lllllll}
(\tau_a & \geq & \tau_s & + & 0) & \wedge \\
(\tau_a & \leq & \tau_s & + & 1) & \wedge \\
(\tau_b & \leq & \tau_s & + & 3) & \wedge \\
(\tau_b & \leq & \tau_a & + & 1) & \wedge \\
(\tau_c & \geq & \tau_a & + & 0) & \wedge \\
(\tau_c & \geq & \tau_b & + & 0) & \wedge \\
(\tau_d & \leq & \tau_a & + & 1) & \wedge \\
(\tau_d & \leq & \tau_b & + & 1) & .
\end{array}$$

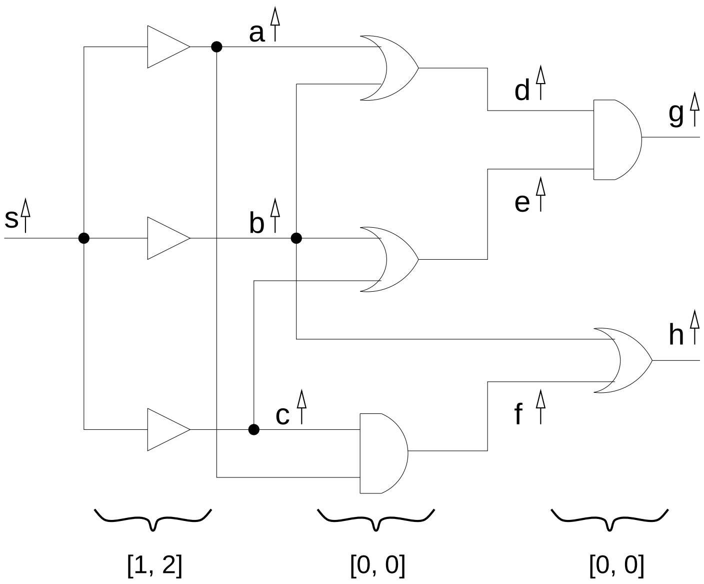
This system admits the solution " $\tau_s = \tau_a = \tau_b = \tau_c =$ any large positive number, and $\tau_d = 0$". Therefore, an upper bound on $(\tau_b - \tau_d)$ subject to this system of constraints is $+\infty$. In reality, however, the maximum achievable bound of $(\tau_b - \tau_d)$ in Fig. 4.1 is 1.

To understand what went amiss, observe that $\tau_d = min(\tau_a + [0,1],\ \tau_b + [0,1])$ yields the system of inequalities: $(\tau_d \leq \tau_a + 1) \wedge (\tau_d \leq \tau_b + 1) \wedge ((\tau_d \geq \tau_b + 0) \vee (\tau_d \geq \tau_b + 0))$. With τ_d set to 0 and τ_b and τ_a set to a large positive number, the disjunctive constraints are violated. This shows that it is important not to disregard the disjunctive constraints in the system of inequalities (4.2). The proposed algorithm takes these into account by means of the approximations described in Section 4.4. As a result, it is able to compute the exact bound from d to b ($\Delta(d, b) = 1$), as shown in Table 4.1.

4.6 Pathological Example

This section describes a pathological example for which the AcyclicApproxSep algorithm fails to compute all pairs of time separations exactly.

Consider the combinational circuit and input stimulus shown in Fig. 4.6 – an $\uparrow$ indicates a rising transition and a $\downarrow$ indicates a falling transition. Gate delays are assumed to vary within the intervals shown, and wires are assumed to have zero delays. Fig. 4.7 shows the timing constraint graph for this system – vertices represent signal transitions and edge delays represent the delays of gates. The Δ matrix computed by applying the



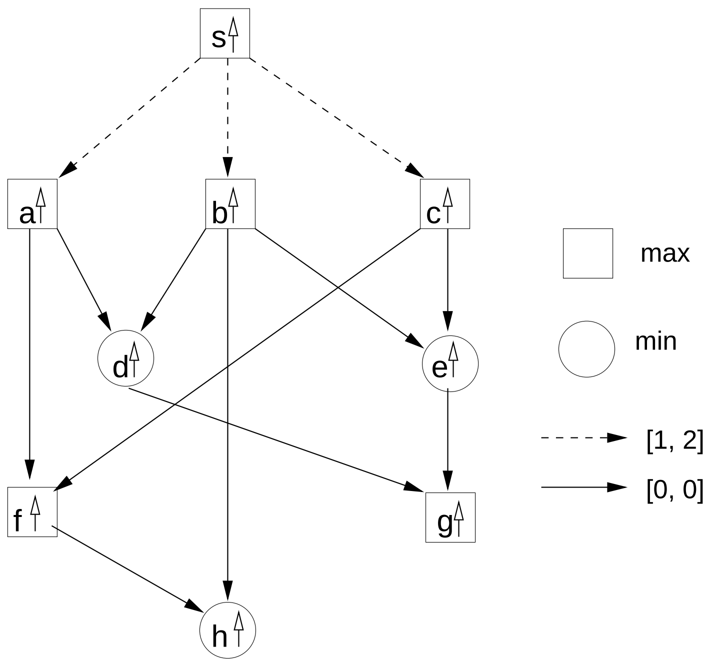
Gate delays are [min, max]. Wires have zero delay.

Figure 4.6: Pathological example for time separations analysis.

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

For clarity, the ↑s have been omitted from the row and column headings.

Table 4.2: Δ matrix for pathological example.



Edge delays are as shown in the legend.

Figure 4.7: Timing constraint graph for pathological example.

AcyclicApproxSep algorithm to this system is shown in Table 4.2.

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

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

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

4.7 Experimental Results

The AcyclicApproxSep algorithm has been implemented and applied to several system specifications. The benchmarks include cyclic timing constraint graphs from [48] and [79] unfolded to various degrees. These are named "hr-n" and "mr-n" in Table 4.3, where different values of r indicate different cyclic graphs, and n denotes the degree of unfolding. The benchmark"mcmill" represents the timing characteristics of a CPU communicating with an EPROM, and has been adapted from McMillan and Dill's paper [72]. The "diffeq" benchmark was obtained from an analysis of an asynchronous differential equation solver chip [100, 22]. The benchmarks labeled “Sat-n” were obtained by generating random 3-

Example Name	# Events (n)	# Min constr.	# Max constr.	Time (s) AcyclicApproxSep	Time (s) [72]	Accuracy
h1-64	321	0	316	0.49	1.48	1.0
h1-256	1281	0	1276	8.65	26.31	1.0
h1-512	2561	0	2556	36.69	113.77	1.0
h2-64	385	0	379	0.75	2.21	1.0
h2-256	1537	0	1531	13.17	40.31	1.0
h2-512	3073	0	3067	55.63	171.86	1.0
m1-64	641	0	255	1.55	4.97	1.0
m1-256	2561	0	1023	27.87	92.05	1.0
m1-512	5121	0	2047	120.69	417.89	1.0
mcmill	13	1	4	< 0.01	0.01	1.0
diffeq	288	4	42	0.21	12309.3	1.0
Sat-3	20	4	9	< 0.01	0.9	0.997
Sat-6	36	7	16	< 0.01	85.81	0.999
Sat-8	44	9	18	< 0.01	287.42	1.0
c1355	948	252	285	2.78	–	N/A
c7552	5258	1174	1169	133.14	–	N/A
c6288	4189	1672	1684	65.38	–	N/A
c3540	3162	417	413	33.31	–	N/A

Min constr. : Number of min constraints with 2 or more arguments
Max constr. : Number of max constraints with 2 or more arguments
Times shown are on MIPS R5000, 150 MHz machine with 128 MB memory.
Entries with “–” indicate that the algorithm did not terminate within 24 hours.
Entries with “N/A” indicate unavailability of the corresponding number.

Table 4.3: Computing **all pairs** of separations.

SAT formulae in n variables and then transforming them to timing constraint graphs using the technique outlined by McMillan and Dill [72]. The other benchmarks were obtained as systems of timing constraints during timing simulation of ISCAS85 circuits with randomly generated input vectors. For each benchmark, Table 4.3 reports the number of events (n), and the number of min and max constraints with two or more arguments. This gives a basis for comparing the run times of the AcyclicApproxSep algorithm with those of [17].

To quantify the accuracy of the computed results, the accuracy of the algorithm is defined as the fraction of all n^2 time separations that is computed exactly. The exact time separation bounds were obtained using the algorithm of McMillan and Dill [72], whenever it terminated successfully within 24 hours. Comparing the exact bounds with those computed by the proposed algorithm indicates that the results have a fairly high degree of accuracy.

The "Sat-n" examples, in which there is some loss of accuracy, are timing constraint graphs representing *unsatisfiable* 3-SAT formulae in n variables. Table 4.3 also shows

Benchmark Name	# Min/max functions	Time (s) for **one objective function** on DEC5000 ([17])
s444	30	0.574
s1423	74	15.5
s9234	227	105.9
s38584	1452	1965.7

Table 4.4: Optimizing **one** linear function by Burks and Sakallah's algorithm [17].

the time taken by the AcyclicApproxSep algorithm to analyze each benchmark on a MIPS R5000 processor running at 150 MHz with 128 MBytes of main memory. These times do not include the time to read in the timing constraint graph and topologically sort the events, which are not significant. For max-only problems, the performances of the AcyclicApproxSep algorithm and McMillan and Dill's algorithm [72] seem comparable. The larger run-times of McMillan and Dill's algorithm are probably due to a non-optimal implementation. However, for systems with both min and max functions, the run times of the AcyclicApproxSep algorithm are tens to thousands of times smaller than the exact branch-and-bound technique of McMillan and Dill [72]. The run times of the AcyclicApproxSep algorithm may also be compared with those of Burks and Sakallah's algorithm [17]. Unfortunately, a direct comparison is not possible because the benchmarks used in [17] were not available for analysis, and the computing platform used in their experiments is different from the one used here. Nevertheless, Table 4.4 lists the run times for some of the largest examples analyzed in [17]. An inspection of the CPU times in Tables 4.3 and 4.4 indicates that for systems of comparable sizes, the AcyclicApproxSep algorithm computes **all $\mathbf{n^2}$** separations in much less time than that required by the algorithm of [17] to compute the maximum value of **one objective function**. Note, however, that Burks and Sakallah's algorithm is more powerful since it is capable of optimizing a general linear objective function, rather than just the difference between two time variables.

4.8 Summary

This chapter described a new algorithm for computing conservative bounds on the time separation of events in systems without choice or repeated events. Unlike several previous approaches, the proposed algorithm is capable of approximately analyzing systems with *both* min and max type constraints in time polynomial in the number of events. The computed bounds are exact if there exists one source event and the timing constraints are either max-only or min-only. For more general systems with multiple source events and both min and max constraints, the bounds are conservative in the worst case, but fairly accurate in practice. Although the proposed algorithm applies *per se* only to systems without repeated events, it can be used as a subroutine in an algorithm for analyzing systems with repeated events, as described in the next chapter. The results of such an analysis are extremely useful in the analysis, optimization and verification of asynchronous and concurrent systems.

Chapter 5

Time Separation of Events: Cyclic Systems

5.1 Introduction

The applicability of the algorithm in Chapter 4 is restricted to systems representable by finite acyclic timing constraint graphs. However, there are several interesting systems in which events occur repeatedly, and which are not finitely representable by acyclic timing constraint graphs. For example, consider an asynchronous controller that cycles through a sequence of states, and sequences the operation of functional units in a datapath. If we assume that the controller iterates through its state transition loop indefinitely, then the corresponding operations in the datapath also repeat indefinitely. An interesting question to ask in such a system, with potential applications to scheduling and allocation, is whether two functional units are ever active at the same time. While this can be formulated as a time separation of events problem, the algorithm in Chapter 4 cannot be used to answer the question because the system is not representable by a finite acyclic timing constraint graph. What we need is a technique for finding bounds on the time separation of events in systems with cyclically repeating behavior. Such an algorithm has several other important applications as well, such as scheduling in embedded systems [56, 24], timing verification of interacting controllers [5, 46, 22], performance analysis of concurrent and embedded systems [46, 98, 22], optimization of timed asynchronous circuits [79], etc.

This chapter describes a polynomial-time algorithm for computing *conservative* bounds on the time separation of events in a class of practical systems with cyclically repeating behavior, called *tightly-coupled systems*. Tightly-coupled systems are a class of choice-free systems with bounded component delays, and may have both min and max type timing constraints. Although the restriction to choice-free systems is significant, there exists a large number of interesting applications that can be modeled and analyzed as choice-free systems. The presence of min constraints makes tightly-coupled systems more general than those considered previously by Amon et al [3], Myers and Meng [79] and Hulgaard et al in [49]. The proposed algorithm is based on the same idea as the algorithm of Myers and Meng [79]; nevertheless there are important extensions to and differences from their work. Although the results are conservative in the worst case, experiments indicate that they are fairly accurate in practice.

The proposed algorithm consists of two phases. In the first phase, the long-term behavior of the system is analyzed by an iterative procedure. This produces conservative bounds on the time separation of events in the long-run that apply regardless of the initial behavior of the system. In the second phase, the initial behavior of the system is analyzed as it starts operating after being powered up. Each phase runs in time polynomial in the size of the timing constraint graph.

The algorithm has been applied for timing verification and performance analysis of an asynchronous differential equation solver chip [101]. The results indicate how variations in component delays affect the performance and correct operation of the chip. The analysis found a known problem in an earlier version of the design in less than 2 seconds on a MIPS R5000 running at 150 MHz. In comparison, the same bug was detected by extensive SPICE simulations only after several hours. This demonstrates the practical utility of the algorithm.

The remainder of this chapter is organized as follows. Section 5.2 reviews related work on time separations analysis of systems with cyclically repeating behavior. Section 5.3 describes cyclic timing constraint graphs for representing timing constraints in systems with cyclically repeating behavior, and characterizes tightly-coupled systems. The time separation of events problem for tightly-coupled systems is also formalized in this section. Section 5.4 presents a polynomial-time algorithm for computing approximate bounds on the time separation of events in tightly-coupled systems. Section 5.5 describes the modeling and analysis of an asynchronous differential equation solver using the proposed algorithm. Finally, the contributions of the chapter are summarized in Section 5.6.

5.2 Related Work

In his Ph.D. thesis [18], Burns used *event-rule (ER)* systems to model asynchronous circuits with fixed component delays and max constraints (Burns called them "conjunctive constraints"). He defined the *cycle period* of the system as the asymptotic average time separation between consecutive occurrences of the same event, and proposed a polynomial-time algorithm for computing this performance metric in ER systems. Burns' algorithm is based on the primal-dual technique for solving special-case linear programs.

Lee [63] extended the work of Burns, and proposed *extended event-rule (XER)* systems to model asynchronous circuits with disjunctive constraints (a generalization of min constraints) in addition to max constraints. His algorithm determines the period of XER systems with fixed component delays, and has worst case exponential complexity. Several other researchers have also proposed techniques for computing performance metrics of asynchronous circuits modeled as systems of repeating events with different assumptions about component delays. A few representative work are those of Williams [94], Nielsen and Kishinevsky [81], Tofts [85], Kudva [57] and Xie and Beerel [96].

In [42, 41], Gunawardena analyzed systems of min and max functions in a theoretical framework, and gave a mathematical characterization of their periodic behavior. He proposed a theory of min-max functions as a non-linear extension of the max-plus algebra, defined a notion of the period of such functions and gave an analytical formula for the period. His treatment in [42, 41] did not consider the possibility of bounded intervals as arguments of the min-max functions. Nevertheless, his theory is useful and provides a sound analytical base for understanding the dynamical behavior of asynchronous circuits with fixed delays.

Hulgaard, Burns, Amon and Borriello [3, 4, 48, 49] proposed the *TSE* algorithm for computing exact bounds on the time separation of events in cyclic systems with bounded component delays but only max constraints. They used *cyclic process graphs* – an extension of Burns' event-rule systems [18] – to represent temporal dependencies between events, and used a functional decomposition technique to analyze implicitly infinite unfoldings of the cyclic graph. Specifically, they cast the problem of computing the maximum separation between a pair of events over infinite unfoldings as a matrix closure computation in a closed semiring. It has been shown in [3, 49] that the TSE algorithm is capable of analyzing several small to medium sized systems in reasonable CPU time. However, the worst case complexity of the algorithm is exponential in the size of the system. In [47], Hulgaard and Burns extended the TSE algorithm to handle systems with *data-independent communication* and either *unique choice* or *extended free choice*, in addition to max constraints. Although their algorithm, called *CTSE*, computes exact bounds, its worst case complexity is exponential in the size of the system. In his Ph.D. thesis [46], Hulgaard also described another type of choice, called *arbitration choice*, that is more general than unique choice or extended free choice. Arbitration choice can be used to model min type timing constraints. However, Hulgaard's algorithm gives *conservative* bounds when arbitration choice is present, in spite of its worst case exponential complexity. More recently, an extension of the TSE algorithm to analyze cyclic systems with both max and linear constraints has been proposed by Jin, Cerny and Hulgaard [53]. This algorithm also has worst-case exponential complexity.

In [79, 78], Myers and Meng proposed a polynomial-time algorithm for computing approximate bounds on the time separation of events in systems modeled by *cyclic constraint graphs*. In their representation, component delays are allowed to vary between specified bounds, but all timing constraints are restricted to be of max type. Their algorithm unfolds the cyclic graph into an infinite acyclic graph, and then analyzes a finite subgraph (usually two unfoldings of the cyclic graph) using McMillan and Dill's algorithm for acyclic graphs [72]. During this analysis, conservative assumptions are made about the timing of events in earlier unfoldings. Consequently, the computed bounds are conservative approximations of the true bounds over all possible unfoldings. They also perform a separate analysis of an initial finite segment of the unfolded graph to determine time separation of events during the initial startup phase of the system.

The algorithm presented in this chapter is based on the same idea as Myers and Meng's algorithm. However there are important extensions to and differences from their work. These are summarized below.

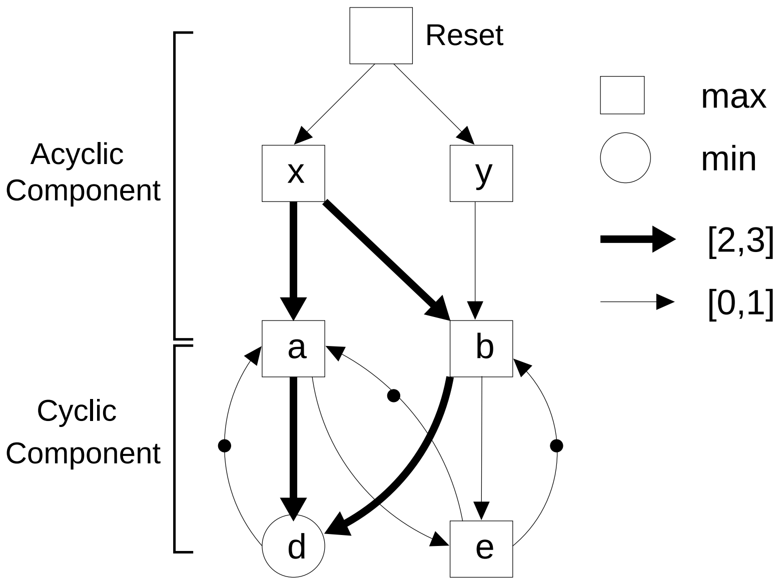
- The proposed algorithm is applicable to tightly-coupled systems with min and/or max type constraints. In contrast, Myers and Meng's algorithm is applicable to max-only systems.
- It is shown that *finite, conservative bounds* on the time separation of events in the long run can be obtained even when the initial behavior of the system is unknown. Although this applies to Myers and Meng's algorithm as well, they do not prove that the bounds are finite [79, 78].
- The proposed algorithm is iterative, and it is shown that the computed time separation bound between every pair of events *monotonically* approaches its asymptotic value with every iteration. Myers and Meng do not address this issue.
- A notion of *convergence* of analysis is defined and an analytic bound on the number of iterations required to converge is provided for the case where all delay bounds are finite, non-negative integers. The work of Myers and Meng lacks such an analytical bound.

5.3 Problem Representation and Formalization

This section describes cyclic timing constraint graphs as a formalism for representing timing constraints in systems with cyclically repeating behavior. A class of practical systems, called tightly-coupled systems, is characterized, and the problem of computing time separation of events in tightly-coupled systems is formalized.

5.3.1 Cyclic Timing Constraint Graphs

A cyclic timing constraint graph is a directed, labeled graph $G = (V, E)$ used to represent timing constraints in systems with cyclically repeating behavior. In general, the graph has two components – an acyclic component modeling the initial behavior of the system after it is powered up, and a cyclic component modeling the subsequent cyclically repeating behavior of the system (see Fig. 5.1). A vertex in the acyclic component of G represents a single occurrence of an event, while a vertex in the cyclic component represents infinite occurrences of an event. The notation of Amon et al [3, 49] is used to distinguish between different occurrences of the same event. Specifically, the k^{th} occurrence of event v is called v_k and k is called the *occurrence index* of v_k. By convention, the first occurrence of v is called v_1. Every vertex is also 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. As in Chapter 4, a vertex associated with a min operator is depicted by a circle while one associated with a max operator is depicted by a square (see Fig. 5.1). It is assumed that there exists a unique Reset event that represents the start of operation of the system. This is represented by a vertex with no incoming edges (source vertex) in the timing constraint graph. For consistency of notation, a vertex is referred to as an event in the remainder of this chapter. An edge $\langle u, v\rangle$ from u to v represents a timing constraint between



Edge delays are as shown in the legend.

Figure 5.1: Cyclic timing constraint graph.

events u and v. As with acyclic timing constraint graphs, the edge $\langle u, v\rangle$ is labeled with a delay $\delta_{u,v}$ that lies within given lower and upper bounds. 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 •, as shown in Fig. 5.1.

Following the terminology of [3, 49, 79], a timing constraint graph is called *well-formed* if the following conditions are satisfied:

- Every cycle has at least one marked edge. This ensures that no event is deadlocked, waiting for itself for it to occur.
- For every event v in the cyclic component, there must exist at least one cycle with exactly one marked edge that contains v. This ensures that the time of occurrence of v_k is well defined for every occurrence index k.

All cyclic graphs considered in this chapter are assumed to be finite and well-formed. Specifically, the graph in Fig. 5.1 is well-formed.

Let $G = (V, E)$ be a 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 event 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 in G^* from u_i to v_{i+1} for all occurrence indices i. 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. The infinite acyclic graph G^* is called the *unfolded graph* of G.

Example 5.1 *Fig. 5.2 shows the unfolded graph of the cyclic timing constraint graph in*

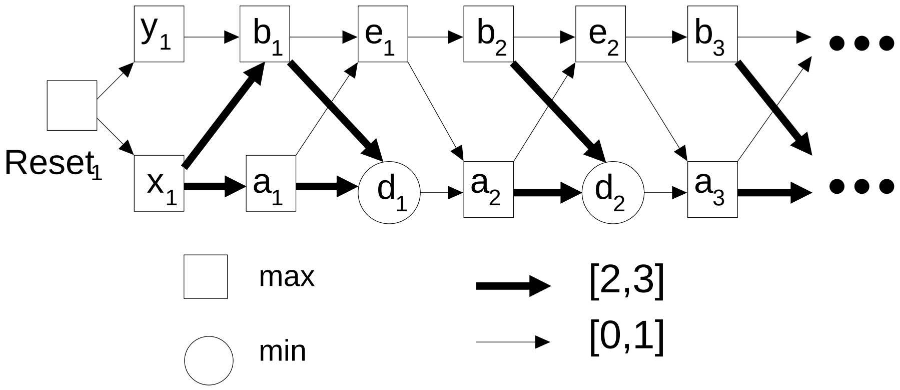


Figure 5.2: Unfolded graph of cyclic timing constraint graph in Fig. 5.1.

Fig. 5.1. Let τ_v represent the time of occurrence of event v. The timing constraints represented by this graph are summarized below.

$$
\begin{array}{lll}
 & \tau_{x_1} = \tau_{\mathsf{Reset}_1} + \delta_{\mathsf{Reset,x}}, & 0 \leq \delta_{\mathsf{Reset,x}} \leq 1 \\
 & \tau_{y_1} = \tau_{\mathsf{Reset}_1} + \delta_{\mathsf{Reset,y}}, & 0 \leq \delta_{\mathsf{Reset,y}} \leq 1 \\
 & \tau_{a_1} = \tau_{x_1} + \delta_{x,a}, & 2 \leq \delta_{x,a} \leq 3 \\
 & \tau_{b_1} = \max(\tau_{x_1} + \delta_{x,b},\ \tau_{y_1} + \delta_{y,b}), & 2 \leq \delta_{x,b} \leq 3,\ 0 \leq \delta_{y,b} \leq 1 \\
\forall k \geq 1, & \tau_{d_k} = \min(\tau_{a_k} + \delta_{a,d}, \tau_{b_k} + \delta_{b,d}), & 2 \leq \delta_{a,d} \leq 3,\ 2 \leq \delta_{b,d} \leq 3 \\
\forall k \geq 1, & \tau_{e_k} = \max(\tau_{a_k} + \delta_{a,e}, \tau_{b_k} + \delta_{b,e}), & 0 \leq \delta_{a,e} \leq 1,\ 0 \leq \delta_{b,e} \leq 1 \\
\forall k \geq 1, & \tau_{a_{k+1}} = \max(\tau_{d_k} + \delta_{d,a}, \tau_{e_k} + \delta_{e,a}), & 0 \leq \delta_{d,a} \leq 1,\ 0 \leq \delta_{e,a} \leq 1 \\
\forall k \geq 1, & \tau_{b_{k+1}} = \tau_{e_k} + \delta_{e,b}, & 0 \leq \delta_{e,b} \leq 1
\end{array}
$$

5.3.2 Tightly-Coupled Systems

This subsection characterizes tightly-coupled systems – a class of cyclic timing constraint graphs that are useful for modeling several practical asynchronous systems. Tightly-coupled systems are more general than the systems considered by Amon et al [3], Myers and Meng [79] and Hulgaard et al in [49]; nevertheless they are amenable to fairly accurate analysis by polynomial-time algorithms.

Consider the unfolded graph $G^* = (V^*, E^*)$ of a well-formed cyclic timing constraint graph. As in Chapter 4, a path from u to v is 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. 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 considered reachable from itself.

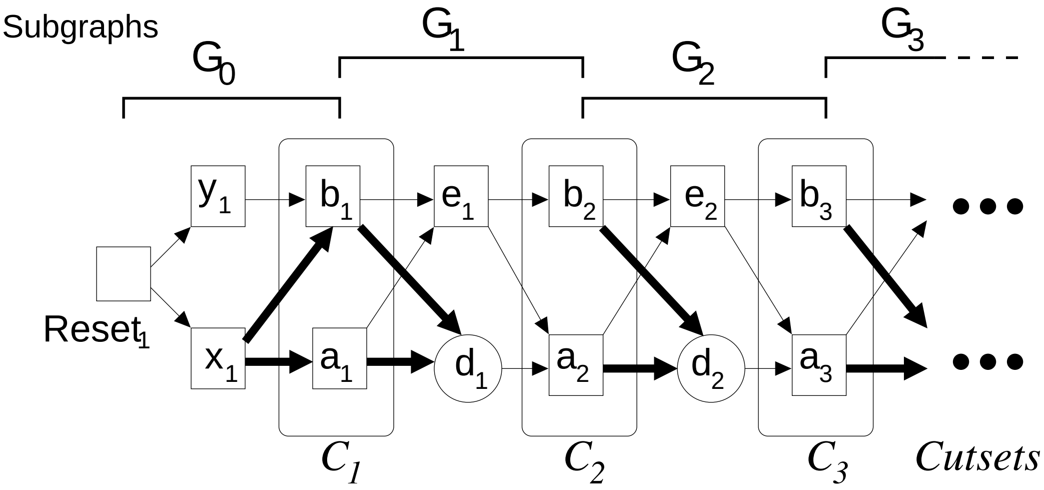


Figure 5.3: Illustrating cutsets and subgraphs of unfolded graph in Fig. 5.2.

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. 5.3 is a cutset of the infinite acyclic graph.

Definition 5.1 *A cyclic timing constraint graph is called **tightly-coupled** if the unfolded graph has a sequence of cutsets, C_i with $i \geq 1$, 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. 5.2, redrawn in Fig. 5.3 for convenience. The circled sets of events, C_1, C_2 and C_3, represent the first three cutsets in an infinite sequence satisfying properties **P1** through **P4**. Consequently, the cyclic timing constraint graph in Fig. 5.1 is tightly-coupled. Note that property P4 allows events with min constraints, such as d_i in Fig. 5.3, to be present as long as a max-only path exists from every u in C_i to every v in C_{i+1}.

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,

$$\begin{aligned} V_i &= \mathcal{R}(C_i) \cap (\mathcal{R}'(C_{i+1}) \cup C_{i+1}) \\ E_i &= \{\langle u, v\rangle : u, v \in V_i \text{ and } \langle u, v\rangle \in E^*\}. \end{aligned}$$

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.,

$$\begin{aligned} V_0 &= \mathcal{R}'(C_1) \cup C_1 \\ E_0 &= \{\langle u, v\rangle : u, v \in V_0;\ \langle u, v\rangle \in E^*\}. \end{aligned}$$

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. Therefore, the unfolded graph is isomorphic to G_0 followed by infinite instances of G_1. Note that properties **P1** and **P2** ensure that all subgraphs G_i for i greater than or equal to 0 are finite.

If α in property **P2** is chosen to be greater than 1 (an example is given later in Fig. 5.6), each subgraph G_i may include multiple occurrences of the same event. This can lead to confusion about the labeling of events in the different subgraphs. To keep the notation unambiguous, the events in the unfolded graph are assumed to be 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 the value of α in property **P2** reduces to 1.

Well-formed cyclic constraint graphs, analyzed by Myers and Meng [79], and *well-formed process graphs*, analyzed by Amon, Hulgaard, Burns and Borriello [3, 49], are special cases of tightly-coupled systems. An *initially marked* rule in Myers and Meng's representation, or an edge with an *occurrence index offset* of 1 in Amon et al's representation corresponds to a marked edge in a tightly-coupled system. If edges with occurrence index offsets greater than 1 are present in the representation of [3, 49], the cyclic process graph can be unfolded a finite number of times until all occurrence index offsets are reduced to either 0 or 1. Properties **P1** through **P4** in Definition 5.1 are satisfied by the unfolded graphs considered by Myers and Meng [79] and Amon et al [3, 49]. Specifically, property **P4** corresponds to the requirement of strongly connected cyclic graphs in their work. Therefore, the systems analyzed by Myers and Meng and Amon et al are essentially tightly-coupled systems restricted to max type timing constraints.

5.3.3 The Problem

The time separations problem for tightly-coupled systems can now be formulated as follows:
Given subgraphs G_i $(i \geq 0)$ of the unfolded graph,

1. *Find upper bounds on the time separations of all ordered pairs of events in G_0.*

2. *Find upper bounds on the time separations of all ordered pairs of events in G_i, maximized over all $i \geq 1$.*

Since G_0 is a finite acyclic graph, the first subproblem is solved by straightforward application of the AcyclicApproxSep algorithm described in Chapter 4. The second subproblem is the primary focus of this chapter.

5.4 Analysis of Tightly-Coupled Systems

This section describes a polynomial-time algorithm for computing the bounds referred to in the problem formulation above. Since G_1 is isomorphic to G_i for all i greater than 1, the algorithm actually analyzes subgraphs G_0 and G_1. The analysis consists of two phases.

- The first phase starts off assuming no knowledge about the initial behavior of the system leading to the events in cutset C_1. Conservative assumptions are made about the time separations of events in C_1, and algorithm AcyclicApproxSep of Chapter 4 is invoked to obtain bounds on the time separations of all pairs of events in G_1. The repetitive structure of the unfolded graph is then exploited to refine these bounds iteratively. If a total of K iterations are performed, the bounds computed in the final iteration apply to events in *every* subgraph G_i for $i \geq k$, *regardless of the initial behavior of the system leading to events in* C_1.

- In the second phase, a finite acyclic graph, modeling the behavior of the system from Reset to events in cutset C_K is constructed and analyzed using the AcyclicApproxSep algorithm. This yields bounds on the time separation of events in subgraphs G_0 through G_{K-1}.

To obtain upper bounds on the time separation of a pair of events maximized over all $G_i,\ i \geq 1$, the maximum of the corresponding bounds computed for subgraphs G_1 through G_K is determined. Below, each phase of the algorithm is described in detail.

5.4.1 Phase I of Analysis

Given an event v in subgraph G_1, let $\Pi(v)$ denote the set of paths from some event in cutset C_1 to v. The *shortest path length* to v, denoted $l(v)$, is defined as:

$$l(v) \quad = \quad \min_{P \,\in\, \Pi(v)} \left(\sum_{\langle x,y \rangle \text{ along } P} d_{x,y} \right). \tag{5.1}$$

Similarly, the *longest path length* to v, denoted $L(v)$, is defined as:

$$L(v) \quad = \quad \max_{P \,\in\, \Pi(v)} \left(\sum_{\langle x,y \rangle \text{ along } P} D_{x,y} \right). \tag{5.2}$$

Since edge delays in timing constraint graphs correspond to component delays in physical systems, and since the delays of all components in practical systems are non-negative and finite, it is henceforth assumed that all edge delays are non-negative and finite. 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.

Let n denote the number of events in G_1. As in Chapter 4, bounds on the time separation of events are represented using an $n \times n$ matrix, Δ. The entry in the row u and column v, denoted $\Delta(u, v)$, is an upper bound on the time separation from event u to event v. Since subgraph G_1 lies between cutsets C_1 and C_2, the source events of G_1 are elements of 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. However, since the first phase of analysis starts assuming no knowledge about the initial behavior of the system, the entries $\Delta(a_1, b_1)$ must be conservatively set to $+\infty$. In general, this can lead to the computed bounds on separations between other pairs of events in G_1 to be $+\infty$. However, in tightly-coupled systems, these separation bounds are finite.

Theorem 5.1 *For every pair of events u and v in cutset C_2 of a tightly-coupled system, the value of $\Delta(u, v)$ computed by algorithm* AcyclicApproxSep *is bounded above by $L(v) - l(u)$, regardless of the time separations between events in C_1.*

Proof: Let there be m events in C_1. By definition, C_2 also has m events. Without loss of generality, the events in G_1 are assigned topological indices such that events 0 through $m-1$ are in C_1 and events $n-m$ through $n-1$ are in C_2. The theorem is proved by showing that $\Delta(i, j)$ is bounded above by $L(j) - l(i)$ for all i and j in $n-m$ through $n-1$.

Let i be an integer in the above range. In the remainder of the proof, i is assumed to be fixed. The proof consists of three parts. First, it is shown that $\Delta(i, j)$ is bounded above by $L(j) - l(i)$ when j lies in 0 through $m-1$. This is then used to prove that all entries

$\Delta(i, j)$, with j in the range m through $i-1$, are similarly bounded. Finally, it is shown that $\Delta(i, j)$ is bounded above by $L(j) - l(i)$ when j lies in $i+1$ through $n-1$. Note that in the first two cases, $\Delta(i, j)$ is computed in the i^{th} iteration of the outer loop of algorithm AcyclicApproxSep (see Fig. 4.4). However, in the third case, $\Delta(i, j)$ is computed in the j^{th} iteration of the outer loop.

(I): By property **P4** in Definition 5.1, there exists a path P from event j to event i such that all events on P have max type constraints. Let v be the predecessor event of i along P. It follows from step (i) of function ComputeInitialEstimates (see Fig. 4.4) that

$$\Delta(i, j) \quad \leq \quad \Delta(v, j) - d_{v,i}. \tag{5.3}$$

However, event v itself lies on path P. Since the events are topologically indexed, this implies that index j is less or equal to than index v. Therefore, $\Delta(v, j)$ is computed in the v^{th}iteration of the outer loop of the AcyclicApproxSep algorithm. Also, since v lies on P, it must be a max type event. Therefore, if u is the predecessor event of v along P, it follows from step (i) of function ComputeInitialEstimates that

$$\Delta(v, j) \quad \leq \quad \Delta(u, j) - d_{u,v}. \tag{5.4}$$

Continuing this argument and adding the resulting inequalities, we obtain:

$$\begin{aligned} \Delta(i, j) \quad &\leq \quad \Delta(j, j) \quad - \sum_{\langle x,y \rangle \text{ along } P} d_{x,y} \\ &\leq \quad -l(i) \qquad (5.5) \\ &\leq \quad L(j) - l(i). \qquad (5.6) \end{aligned}$$

Inequality (5.5) follows from the observation that $\Delta(j, j)$ is 0 by definition, and inequality (5.6) follows from the observation that $L(j)$ is non-negative.

(II): Now, consider $\Delta(i, j)$, with j lying in the range m through $i-1$. The proof is by complete induction on j.

Basis: From part 1, it follows that for all j in 0 through $m-1$, the entry $\Delta(i,j)$ is bounded above by $L(j) - l(i)$.

Hypothesis: Let j be an integer in m 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

$$\begin{aligned}\Delta(i,j) &\leq \max_{k \,\in\, preds(j)} (\Delta(i,k) + D_{k,j}) \\ &\leq \max_{k \,\in\, preds(j)} (L(k) - l(i) + D_{k,j}) \qquad (5.7) \\ &\leq L(j) - l(i). \qquad (5.8)\end{aligned}$$

Inequality (5.7) follows from the hypothesis and inequality (5.8) follows from the observation that the longest path length to j is at least as large as the longest path length to any predecessor of j plus the maximum delay of the corresponding edge to j.

(III): Finally, consider $\Delta(i,j)$, with j lying in the range $i+1$ through $n-1$. As noted above, the entry $\Delta(i,j)$ is computed in the j^{th} iteration of the outer loop of the AcyclicApproxSep algorithm, since index j is greater than index i. The proof is by complete induction on j.

Basis: From parts **(I)** and **(II)**, it follows that for all j in 0 through $i-1$, the entry $\Delta(i,j)$ is bounded above $L(j) - l(i)$. In addition $\Delta(i,i)$, which is 0 by definition, is bounded above by $L(i) - l(i)$, which is non-negative by definition.

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

$$\Delta(i,j) \leq \max_{k \,\in\, preds(j)} (\Delta(i,k) + D_{k,j}) \qquad (5.9)$$

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

This completes the proof of the theorem. ■

Corollary 5.1 *For every pair of events u and v in cutset C_2 of a tightly-coupled system, the value of $\Delta(u, v)$ computed by algorithm* AcyclicApproxSep *is finite.*

Proof: Follows from Theorem 5.1 and the observation that $\Delta(i, j)$ cannot be $-\infty$ because the Δ matrix is initialized with $+\infty$ and finite values to start with, and subsequently, only finite edge delay bounds are added or subtracted a finite number of times to obtain the final values of the entries. ■

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. Although the algorithm actually analyzes subgraph G_1 in all the iterations, 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 the i^{th} iteration are stored in an $n \times n$ matrix, Δ_i. The first iteration starts with no knowledge about the past behavior of the system. This is equivalent to 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 5.1, the computed separation 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 is re-invoked to compute bounds on the time separations of all pairs of events in G_1. The intuition behind this 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 assigned 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_1 are then computed in iteration $i + 1$ iteration 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. In this case, the input conditions for iteration $i + 1$ are identical to those for iteration i iteration, so additional iterations are not meaningful.

CyclicApproxSep (C_1, C_2: cutsets; G_0, G_1: acyclic subgraphs; K_{max}: integer)

Phase I: /* Analysis of G_1 */
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: /* Analysis of G_0 through G_{K-1}*/
1. Construct $G_{startup}$, the acyclic graph composed of G_0 followed by
   $K - 1$ instances of G_1.
   /* $G_{startup}$ is isomorphic to G_0 through G_{K-1} considered together. */
2. **for all** ordered pairs of events (a, b) in $G_{startup}$
      $\Delta(a, b) = +\infty$;
3. AyclicApproxSep($G_{startup}, \Delta$);

AcyclicApproxSep (G: acyclic graph)
   Same as algorithm in Fig. 4.4.

Figure 5.4: Timing analysis algorithm for tightly-coupled systems.

(b) The number of iterations reaches a user-specified upper bound, K_{max}.

If condition (a) is satisfied, the iterative analysis is said to have *converged* to a fix-point. If all edge delay bounds are integers, an upper bound on the number of iterations required to converge has been derived. However, in practice, convergence usually occurs within a small number of iterations – at most 5 in the experiments that have been performed. The specific example which required 5 iterations is described later. Condition (b) ensures that the analysis terminates after a constant number of iterations if it has not already converged by then. The pseudo-code in Phase I of Fig. 5.4 describes the steps of the iterative algorithm outlined above. The jump to Phase II in the "if" part of step $2(b)(ii)$ is explained later in Section 5.4.2.

If there are n events in subgraph G_1 and p denotes the maximum number of predecessors of an event, the complexity of algorithm AcyclicApproxSep, as noted in Section 4.5 is $O(n^2.p)$. The complexity of Phase I is therefore, given by $O(K_{max}.n^2.p)$.

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

Lemma 5.1 *For all iteration indices i in Phase I, $\Delta_{i+1} \leq_{\mathbf{M}} \Delta_i$. This property is called* ***monotonic convergence*** *of the time separation intervals.*

Proof: To prove the lemma, we first make the following simple observation about algorithm AcyclicApproxSep (see Fig. 4.4).

Observation 5.1 *Since all expressions on the right hand side of the assignments labeled (i) through (iv) and (2a) through (2d) in Fig. 4.4 involve non-negated Δ matrix entries, the bounds computed by algorithm* AcyclicApproxSep *cannot increase if all edge delay bounds are unchanged and the bounds between every pair of source events are not increased.*

The lemma is now proved by induction on i.

Basis $(i = 1)$: In the first iteration, $\Delta_1(a_1, b_1)$ is set to $+\infty$ for every pair of events a_1 and b_1 in C_1. In the second iteration, $\Delta_2(a_1, b_1)$ is set equal to $\Delta_1(a_2, b_2)$, where a_2 and b_2 are

the corresponding events in C_2. However, by Corollary 5.1, $\Delta_1(a_2, b_2)$ is finite. It follows from Observation 5.1 that $\Delta_2 \leq_{\mathbf{M}} \Delta_1$.
Hypothesis: For i greater than or equal to 1, let $\Delta_{i+1} \leq_{\mathbf{M}} \Delta_1$.
Induction: By hypothesis, $\Delta_{i+1}(a_2, b_2)$ is no larger than $\Delta_i(a_2, b_2)$ for every pair of events a_2 and b_2 in C_2. However, by step 2(b)(iii) of Phase I in Fig. 5.4, $\Delta_{i+1}(a_2, b_2)$ equals $\Delta_{i+2}(a_1, b_1)$ and $\Delta_i(a_2, b_2)$ equals $\Delta_{i+1}(a_1, b_1)$. Therefore, $\Delta_{i+2}(a_1, b_1)$ is no larger than $\Delta_{i+1}(a_1, b_1)$, for every pair of events a_1 and b_1 in C_1. The proof then follows from Observation 5.1. ■

Theorem 5.2 *Suppose Phase I terminates after K iterations and K is at least 2. The bounds computed in the final iteration are finite and apply to events in every subgraph G_i for $i \geq K$, regardless of the initial behavior of the system.*

Proof: The proof that the bounds are finite follows from the observation that if all edge delays and separations between source events are finitely bounded, the bounds computed by algorithm AcyclicApproxSep are finite. If at least two iterations are made, Corollary 5.1 and step 2(b)(iii) of Phase I in Fig. 5.4 ensure that the bounds between every pair of source events in C_1 are finite.

The remainder of the theorem is proved by inductively showing that the bounds computed in iteration i apply to all events in G_i. Since $\Delta_i \leq_{\mathbf{M}} \Delta_K$ for all $i \geq K$ (by Lemma 5.1), the theorem follows.
Basis $(i = 1)$: The first iteration is viewed as analysis of G_1. Since $\Delta_1(a_1, b_1)$ is set equal to $+\infty$ for every a_1 and b_1 in C_1, the value of $\Delta_1(a_1, b_1)$ is a true upper bound on the separation from event a_1 to event b_1, regardless of the initial behavior of the system. Therefore, by Theorem 4.1 of Chapter 4, $\Delta(u_1, v_1)$ is an upper bound on the 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 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. Note that G_i is "sandwiched" between cutsets C_i and C_{i+1}.

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 assigned to $\Delta_i(a_1, b_1)$ at the end of iteration $i-1$, the value of $\Delta_i(a_1, b_1)$ is a true upper bound on the time separation from a_i to b_i. The proof then follows from Theorem 4.1 of Chapter 4. ■

Lemma 5.2 *Let $l(v)$ and $L(v)$ be the shortest and longest path lengths to v from any event in C_1. If cutset C_2 has m events, and all edge delay bounds are integers, Phase I converges in at most $(2 + (m-1) \cdot \sum_{v \in C_2}(L(v) - l(v)))$ iterations.*

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 the number of iterations in between is bounded above by $(m-1) \cdot \sum_{v \in C_2}(L(v) - l(v))$

First, note that Theorem 4.1 of Chapter 4 implies that for every pair of events a and b in G_1,

$$-\Delta(b, a) \quad \leq \quad \tau_b - \tau_a \quad \leq \quad \Delta(a, b).$$

Therefore, $\Delta(a, b)$ is bounded below by $-\Delta(b, a)$.

From Theorem 5.1, 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 5.1 implies that in every subsequent iteration, the value of $\Delta(u, v)$ either remains constant or decreases. The minimum value to which it can decrease is $-\Delta(v, u)$, which, in turn, 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 $(m-1) \cdot \sum_{v \in C_2}(L(v) - l(v))$ 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. ■

The bound given by Lemma 5.2 can be large for certain edge delays. In practice, however, the analysis usually converges within a few iterations. By choosing K_{max} to be in the range $10 - -20$ regardless of the graph being analyzed, the analysis is expected to converge in most practical systems, while the complexity remains bounded by a polynomial in the system size.

5.4.2 Phase II of Analysis

Suppose Phase I terminates after K iterations. By Theorem 5.2, the bounds in Δ_K apply to events in every subgraph G_i for $i \geq k$. In order 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 5.4.

Let there be q events in subgraph G_1 and n events in subgraph G_0. The complexity of constructing the acyclic subgraph consisting of G_0 through G_K is $O(q^2 + K_{max}.n^2)$. The resulting graph has at most $q + K_{max}.n$ events. If p denotes the maximum number of predecessors of an event, algorithm AcyclicApproxSep takes $O((q + K_{max}.n)^2.p)$ time to analyze this graph. Therefore, the complexity of Phase II is $O((q + K_{max}.n)^2.p)$.

5.4.3 Examples

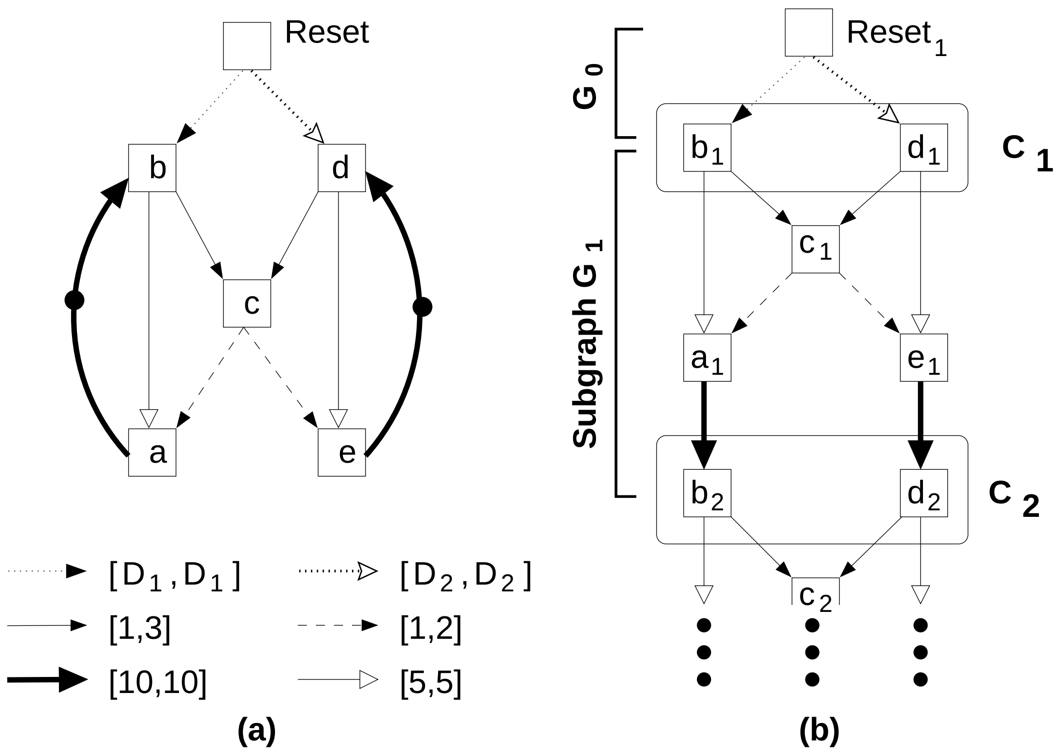
Let us now apply the above algorithm to two interesting examples, adapted from previous publications [3, 48].

Example 5.2 *This example illustrates that conservative bounds on the long-term time separation of events can be obtained without any knowledge of the initial behavior, even when the exact bounds strongly depend on the initial behavior.*

Consider the system shown in Fig. 5.5a. It has been shown by Amon et al [3] and Hulgaard et al [48] that the maximum attainable value of $\tau_{e_i} - \tau_{a_i}$ *depends on the difference between* D_2 *and* D_1*. Their results are summarized in Table 5.1.*

$D_2 - D_1$	≥ 3	$= 2$	$= 1$	≤ 0
Max. attainable $\tau_{e_i} - \tau_{a_i}$	3	2	1	0

Table 5.1: Exact time separation bounds of system in Fig. 5.5.



Edge delays are as shown in the legend.

Figure 5.5: (a) Timing constraint graph of first example [3, 48]; (b) Unfolded graph with cutsets C_1 and C_2, and subgraphs G_0 and G_1.

If we apply the algorithm proposed in this chapter to subgraph G_1 shown in Fig. 5.5b, Phase I converges after 2 iterations and the value of $\Delta_2(a_1, e_1)$ is 3. From Table 5.1, this bound is seen to be exact, considering all possible combinations of D_1 and D_2.

Example 5.3 *The second example, shown in Fig. 5.6a, exhibits unusual behavior; yet, the proposed algorithm is able to compute exact bounds. This example demonstrates that the number of iterations required to converge depends on the values of the edge delay bounds in general.*

Let M^i denote the maximum attainable value of $\tau_{a_i} - \tau_{a_{i-1}}$, for all i greater than or equal to two[1]. It has been shown by other researchers [3, 48] that M_i depends on both i and λ. This dependence is shown in Table 5.2.

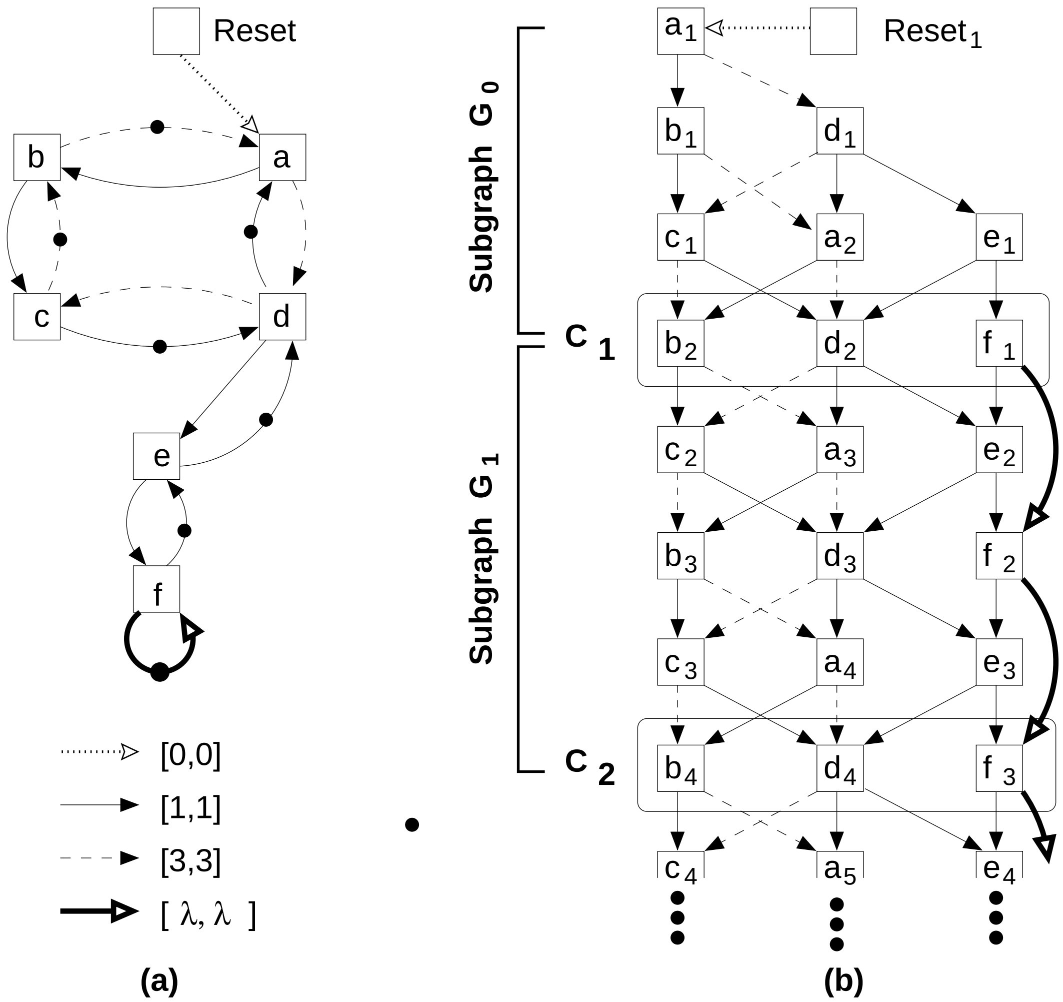
If $\lambda = 6$:		If $\lambda = 9$:								
M^{even}	M^{odd}	M^2	M^3	M^4	M^5	M^6	M^7	M^8	M^9	$M^{\geq 10}$
4	8	4	8	4	8	4	8	8	9	9

Table 5.2: Exact time separation bounds for system in Fig. 5.6. $\tau_{a_i} - \tau_{a_{i-1}} \leq M^i$.

Fig. 5.6b shows subgraph G_1 of the unfolded graph. For clarity, the events in G_1 are not *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.*

If we apply the proposed algorithm, it is found 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. Therefore, 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. Hence, the computed bounds are exact, considering all instances of a_i and a_{i-1}.

[1] M^i in this notation equals Δ^{i-1} in the notation of Amon et al [3] and Hulgaard et al [48].



Edge delays are as shown in the legend.

Figure 5.6: (a) Timing constraint graph of second example [3, 48]; (b) Unfolded graph with cutsets C_1 and C_2 and subgraphs G_0 and G_1.

5.5 Analyzing an Asynchronous Chip

This section describes a practical application of the algorithm proposed in this chapter, namely timing verification and performance analysis of an asynchronous differential equation solver chip. The discussion starts with a brief overview of the chip, henceforth called DiffEq. This is followed by techniques for modeling the temporal behavior of interacting controllers and datapath elements. Performance metrics and timing constraints required for correct operation of the chip are then formulated as time separations between events. Finally, bounds on all required time separations are computed using the proposed algorithm. The entire analysis (Phases I and II) takes less than 2 seconds on a MIPS R5000 running at 150 MHz. The results of such an analysis are useful for studying the effects of component delay variations on the performance and correct operation of the chip. In fact, the analysis found a known timing bug in an earlier version of the design. The same bug had also been detected earlier by extensive SPICE simulations, but only after several hours.

5.5.1 Chip Overview

DiffEq is a hardware implementation of the *forward Euler method* for numerically computing values of y that satisfy the differential equation $y'' + 3xy' + 3y = 0$. The variable x is assumed to vary from $x(0)$ to a in steps of dx, and the initial values of y and y' are assumed to be $y(0)$ and u. Fig. 5.7 shows the pseudo-code of the algorithm [29] implemented by DiffEq.

Fig. 5.8, adapted from Yun et al [101], shows the architecture and dataflow graph for one iteration of operation of DiffEq. The dataflow graph consists of three threads of computation that synchronize across iteration boundaries to resolve data dependencies. The solver iterates through the dataflow graph until the termination condition, $(x < a)$, is satisfied. For simplicity, this discussion, however, focuses only on the behavior of the system as it iterates through the dataflow graph a large number of times, so the termination condition is ignored. Note that as a consequence of this simplification, the results of the analysis, such as timing verification results, may not apply in the final iteration when the system actually terminates the computation.

```
diffeq{
       read (x, y, u, dx, a);
       while (x < a) do {
          x1 = x + dx;
          u1 = u - 3 * dx * (u * x + y);
          y1 = y + u * dx;
          x  = x1;
          u  = u1;
          y  = y1;
       }
       write (y );
}
```

Figure 5.7: Algorithm implemented by DiffEq.

The datapath consists of four units, shown shaded in Fig. 5.8b. Each unit is structurally identical to those typically found in synchronous designs, except for the completion-reporting logic associated with the functional units. There are four functional units – two adder/subtracters, labeled ALU1 and ALU2, and two multipliers labeled MUL1 and MUL2. Each functional unit is implemented in domino logic.

Each shaded datapath unit is controlled by an extended burst-mode (XBM) controller implemented in the 3D design style [99][2]. As in synchronous systems, the controllers are 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 correct functioning of the system. For example, data inputs to domino circuits must stabilize before evaluation begins, and setup, hold-time and minimum clock pulse-width constraints of registers must be satisfied. The controllers also communicate with each other by means of a handshaking protocol which, unlike *speed-independent* or *delay-insensitive* designs, assumes safe timing bounds in signaling. Details of the DiffEq design are described elsewhere [101].

[2]The reader may refer to Chapter 3 for a brief review of XBM controllers and 3D design.

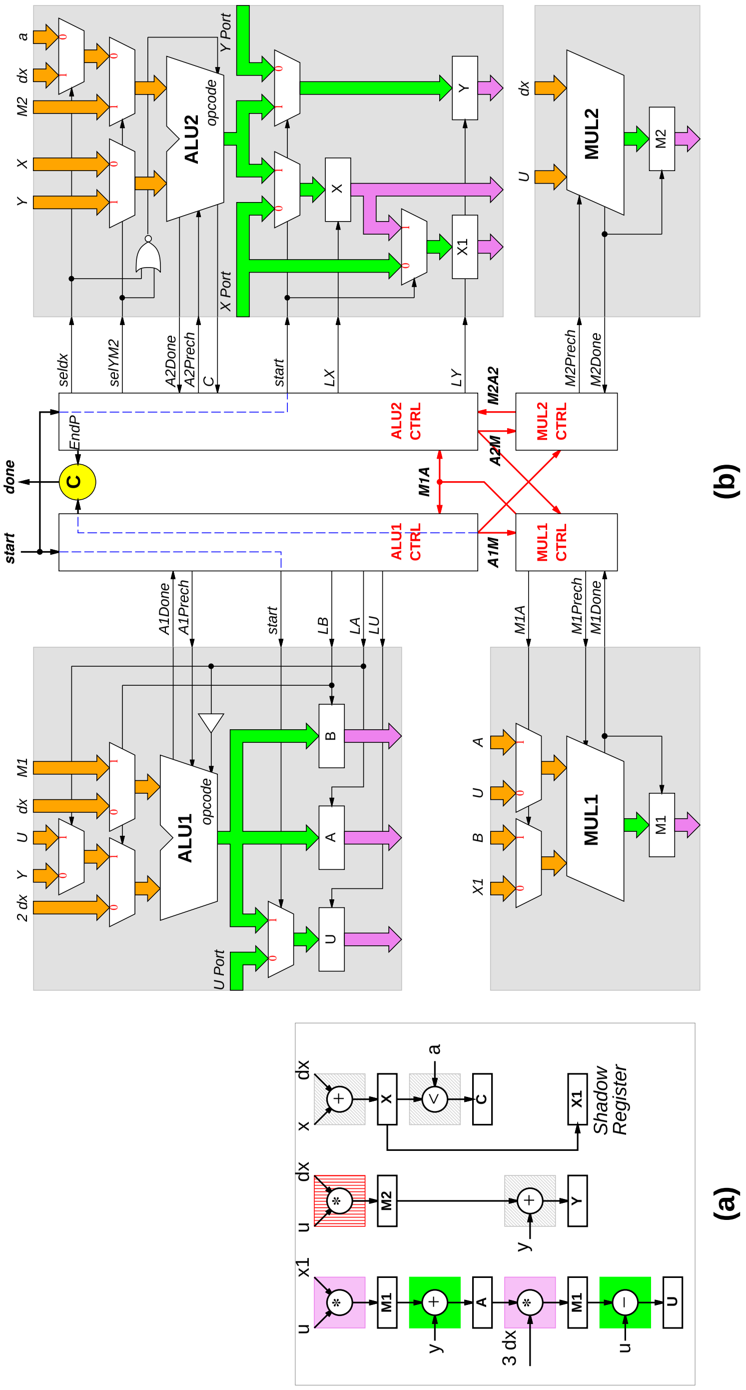


Figure 5.8: (a) Dataflow graph; (b) Architecture of DiffEq, adapted from Yun et al [101].

5.5.2 Modeling Controller Timing

Without the check for the termination condition, each XBM controller in DiffEq cycles through a deterministic sequence of states and exhibits choice-free behavior. Therefore, cyclic timing constraint graphs can be used to model their behavior.

Each controller is viewed 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, such as minimum feedback delays [99], can be obtained by the technique described in Chapter 3, and are assumed to be satisfied.

To model the behavior of a controller, its state transition diagram is first examined. 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 [99] 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 in a timing constraint graph, a max-type event, m, is created, and zero-delay edges are drawn from the events representing the terminating transitions in $\mathcal{I}$ to m. Edges are also drawn from m to each event representing the concurrently enabled transitions in $\mathcal{O}$. These edges are labeled 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. 5.9a 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* [99]. Fig. 5.9b 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. 5.9b 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 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. 5.9b.

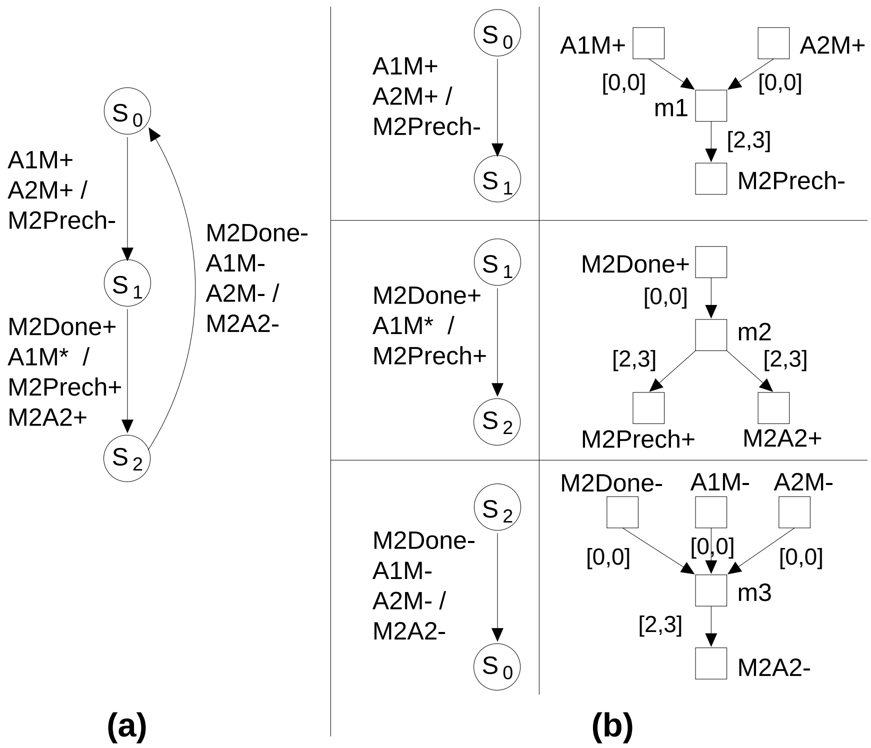
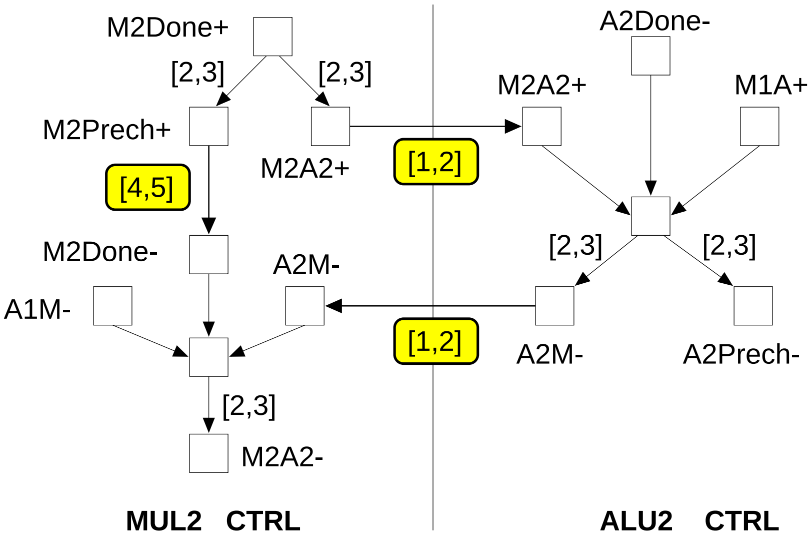


Figure 5.9: Modeling temporal behavior of MUL2 CTRL. (a) XBM state transition diagram; (b) Timing constraint graph fragments (controller delay = $[2, 3]$).



Unlabeled edges are [0,0].
Precharge delay = $[4,5]$; wire delay between ALU2 CTRL and MUL2 CTRL $= [1,2]$.

Figure 5.10: Stitching timing constraint graphs.

The procedure outlined above gives 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. From the perspective of each XBM controller, the added edges essentially correspond to responses from its environment. For example, consider the following sequence of operations: MUL2 CTRL raises M2prech and in response, the MUL2 unit lowers M2done to notify MUL2 CTRL of its status. If the delay from M2prech going high to M2done going high, plus the signal propagation delay from MUL2 to MUL2 CTRL lies between 4 and 5 units, an edge is drawn from M2prech+ to M2done− and labeled with $[4, 5]$, as shown in Fig 5.10. 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 between ALU2 CTRL and MUL2 CTRL, the corresponding events in the timing constraint graphs for ALU2 CTRL and MUL2 CTRL are connected by an edge and labeled with the signal propagation delay from ALU2 CTRL to MUL2 CTRL (see Fig. 5.10). Continuing this process, a timing constraint graph modeling the behavior of all the controllers during one iteration of operation of DiffEq can be constructed.

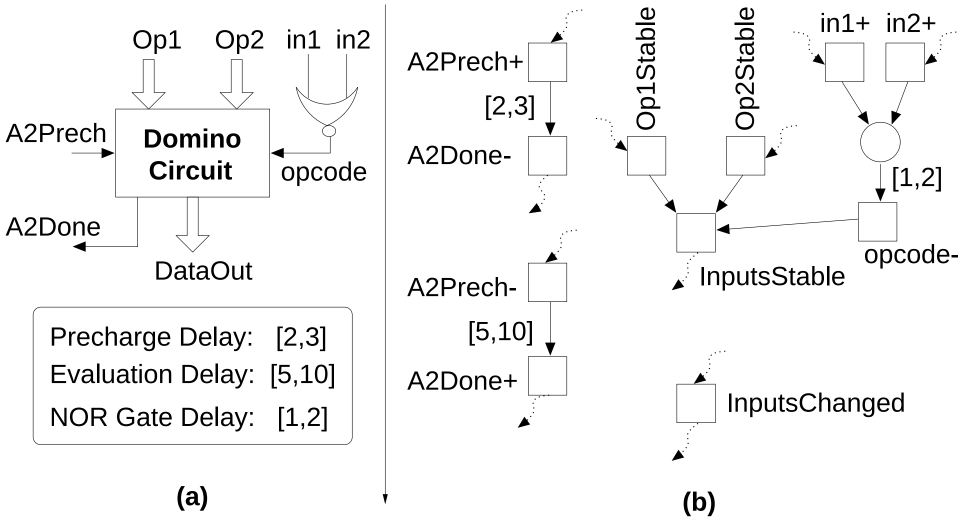
5.5.3 Modeling Datapath Timing

The datapath components used in DiffEq are arithmetic circuits implemented in domino logic, edge-triggered registers and multiplexers. In this subsection, the behavior of each component is modeled with timing constraint graph fragments. Note that different fragments must be used to model different instances of computation by the same unit.

Arithmetic circuits in domino logic

Fig. 5.11a depicts the ALU2 unit in DiffEq. This is an adder/subtracter implemented in domino logic. Different phases of the A2Prech signal are used for precharge and evaluation. The A2Done signal is used to notify other units that all internal nodes have been precharged (A2Done−) or that evaluation has completed (A2Done+). The level of signal opcode determines the type of operation (addition or subtraction) to be performed.

Assuming that both in1 and in2 in Fig. 5.11a have rising transitions, the temporal behavior of the ALU2 unit is modeled in Fig. 5.11b. When the precharge signal is asserted (A2Prech+), 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 and also drives A2Done low. To model this, an edge is drawn from A2Prech+ to A2Done− and labeled with the precharge delay.



Squares represent max events; circles represent min events.
Dotted edges represent connections to/from other events. Unlabeled edges are [0,0].

Figure 5.11: (a) Domino circuit; (b) Timing constraint graph.

The signal opcode at the output of the NOR gate in Fig. 5.11a falls as soon as one of the inputs rises. Therefore, the falling of opcode is modeled as a min type event (circle) in Fig. 5.11b. This clearly 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, while InputsStable represents the stabilization of all inputs (both data operands and opcode). Since InputsStable occurs only after all the inputs have stabilized, it is a max type event and is represented by a square in Fig. 5.11b.

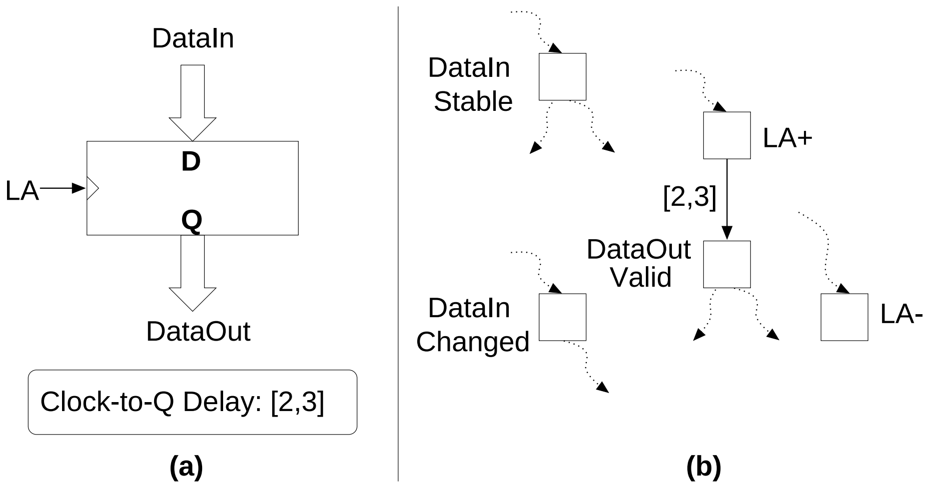
Evaluation begins as soon as A2Prech is de-asserted (A2Prech$-$). Assuming that InputsStable occurs before A2Prech$-$ (this is checked during analysis), the circuit computes the sum (or difference) of the operands and raises A2Done$+$ after some data-dependent computation delay. To model this behavior, an edge is drawn from A2Prech$-$ to A2Done$+$ and labeled with the computation delay. The event InputsChanged in Fig. 5.11b represents a subsequent change in the value of the inputs.

Edge-triggered registers

Fig. 5.12a depicts a positive edge-triggered register clocked by the signal LA. This represents register A in Fig. 5.8b. For clarity, it is assumed that there are no asynchronous set/reset inputs. Assuming the data input remains stable within the setup and hold-time window (this is checked during analysis), input data is loaded when LA rises, and appears at the output port after a "clock-to-Q" delay. Let DataInStable and DataInChanged represent the stabilization of the data input prior to latching and its subsequent change. Similarly, let DataOutValid represent the appearance of the loaded data at the output port. The behavior of the register is then modeled by drawing an edge from LA$+$ to DataOutValid and labeling it with the clock-to-Q delay, as shown in Fig. 5.12b. The event LA$-$ in Fig. 5.12b represents the falling edge of LA after the latching LA$+$ edge. Note that if a level-sensitive latch was used, 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 setup and hold times are satisfied, which can be checked by computing the time separations from DataInStable and DataInChanged to the latching edge, $LA-$.

Multiplexers

Fig. 5.13a depicts a 2-to-1 multiplexer. Depending on the value of Control, either the data at In0 or that at In1 is routed to the Out port. For simplicity, it is assumed that either the Control input stabilizes before the data inputs or vice versa (this can be checked during analysis). If there is a transition on Control, the data at the output is not valid at least until Control has stabilized. Similarly, if a data from a particular input port must be routed to



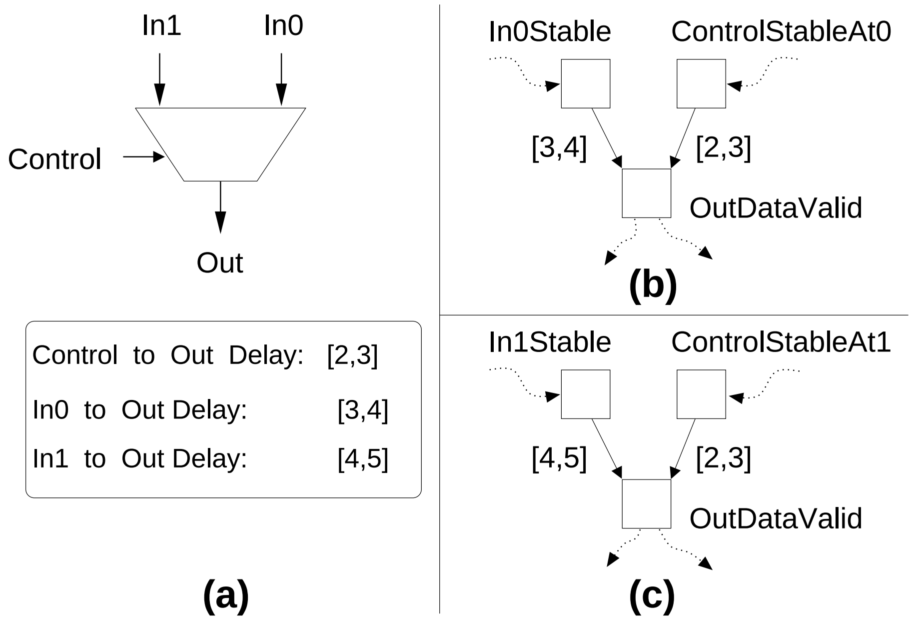
Dotted edges represent connections to/from other events.

Figure 5.12: (a) Edge-triggered register; (b) Timing constraint graph.

the output, the data at the output is invalid at least until the input port has stabilized. This behavior is modeled by the timing constraint graph in Figure 5.13b. The delay from the Control input to the Out port is measured assuming that the data input ports are stable. Similarly, the delay from In0 (In1) to the Out port is measured assuming that Control is stable at 0 (1). Note that this technique assumes that we know in advance the value of the control input that is used, so that the correct graph fragment is used to model the temporal behavior of the system. The above technique is easily extendible to model larger multiplexers, e.g., 4-to-1 multiplexers.

5.5.4 Formulating Performance Metrics and Timing Constraints

Once the timing constraint graph modeling a generic iteration of the dataflow graph has been constructed, performance metrics and timing constraints required for correct operation of the system are formulated as time separations between events.



Dotted edges represent connections to/from other events.

Figure 5.13: (a) 2-to-1 multiplexer; (b) Timing constraint graph if Control stabilizes to 0; (c) Same if Control stabilizes to 1.

Performance metrics

Two interesting performance metrics are considered:

- *Loop delay*: This is 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 of the dataflow graph.
- *Cycle time*: This is the delay between similar events in successive iterations of the dataflow graph.

Note that since the dataflow graph has multiple threads of computation which do not necessarily start simultaneously or end simultaneously, the loop delay is not the same as the cycle time. The inverse of the cycle time gives an estimate of the rate at which successive values of a variable, such as y in the algorithm of Fig. 5.7, are generated. However, if the values of all variables at the end of n iterations of the dataflow graph are needed, then the total time required is $((n-1) \cdot \text{cycle time} + \text{loop delay})$ since the values of all variables may not be computed within $n \cdot \text{cycle time}$ time.

To compute the loop delay, the timing constraint graph modeling one iteration of operation of DiffEq is considered. Since there are multiple parallel threads of computation in the dataflow graph, the starting time of an iteration is represented by taking the minimum start times of all the threads. 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 then give the best case and worst case loop delay of the dataflow graph. Note that both min and max type constraints are needed in this case.

To estimate the cycle-time, the time separation between the start of two consecutive iterations of the dataflow graph is computed. The start of the current iteration has already been modeled above. The start of the next iteration is modeled in a similar manner 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. Computing bounds on the time separation between StartofIter and StartOfNextIter then gives 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.

It may be argued that bounds on the loop delay or cycle time are not representative performance metrics of a system; instead, average loop delay and cycle time are. However, finding bounds on these metrics 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 maximum delay for n iterations, or the minimum rate at which intermediate results are produced.

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. To check register timing constraints, every instance of every register loading data during one iteration of the dataflow graph is considered. Setup-time violations are checked by computing the minimum time separation between the stabilization of data inputs and the latching edge of the clock signal (DataInStable and LA+ in Fig. 5.12b). 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 between the latching edge of the clock and the subsequent change of the input data (LA+ and DataInChanged in Fig. 5.12b). If this is greater than the hold-time requirement, no hold-time violation is possible. Finally, the minimum clock pulse-width is obtained by determining the minimum time separation between the latching edge of the clock and its subsequent opposite transition (LA+ and LA− in Fig. 5.12b). If this exceeds the required minimum clock pulse-width, there is no clock pulse-width violation.

To verify the timing of domino circuits, every instance of every domino circuit performing computation during one iteration of the dataflow graph is considered. The minimum time separation between the stabilization of all inputs and the start of evaluation

(InputsStable and A2Prech− in Fig. 5.11b) is determined. 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, causing incorrect values to appear at the outputs. Note that if all input data lines rise monotonically, the internal nodes will not 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. Another timing check for domino circuits is to see if the data inputs can change prior to the end of evaluation. In Fig. 5.11b, this is equivalent to checking if the minimum separation from A2Done+ to InputsChanged is negative.

Finally, timing constraints in the inter-controller communication protocol are considered. In DiffEq, these constraints arise from XBM requirements. For example, consider the state transition diagram of MUL2 CTRL in Fig. 5.9a. Rising transitions on both A2M and A1M enable the state change from S_0 to S_1 and trigger a falling transition on M2prech. The XBM semantics require that A2M subsequently remain high until the controller has changed state from S_1 to S_2, 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 non-negative. Other timing constraints in the inter-controller communication protocol are of a similar nature, i.e., they are needed to ensure that some event occurs some time after other event. Such timing constraints can always be formulated in terms of time separations between events. 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.

5.5.5 Experimental Results

The complete DiffEq chip has been modeled and analyzed using the techniques described above. Since some time separations of interest involve events in two consecutive iterations of the dataflow graph, cutsets 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 simulation 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 delay bounds.

Subgraph G_1 has 288 events. During Phase I, bounds on the time separations of 129 pairs of events are monitored to check for timing constraint violations and to estimate performance metrics. Phase I converges after 2 iterations and takes 0.38s on a 150 MHz MIPS R5000 machine with 128 MB of memory. In comparison, the symbolic performance analysis method of Xie and Beerel [96] took 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 separations 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 above.

Since both min and max type constraints are present in the timing constraint graph, the bounds computed by algorithm AcyclicApproxSep during Phases I and II may be inexact. Therefore, for comparison, the analysis for $\pm 5\%$ delay variations is repeated with McMillan and Dill's branch-and-bound algorithm [72] used in place of the AcyclicApproxSep algorithm. All the bounds are found to be identical to those obtained with AcyclicApproxSep. However, the complete analysis (Phase I and Phase II) using McMillan and Dill's algorithm takes more than 11 hours on the same computing platform.

Two subsets of timing verification results, corresponding to $\pm 5\%$ and $\pm 10\%$ variations of delays, are presented in Table 5.3. Each entry lists a pair of events, the minimum separation between these events required for correct operation of the system, and the computed bounds on the separation between these events (lower bounds are the minimum of the lower bounds in Phases I and II; similarly upper bounds are the maximum of the upper bounds in the two phases). Table 5.4 shows the loop delay and cycle-time of DiffEq obtained with 0%, $\pm 5\%$ and $\pm 10\%$ variations in delays. The 0% entries give the loop delay and cycle time variations due to variations in datapath delays arising from data dependency. The 5% and 10% entries give the added variation in performance due to small variations in component

	Event1	Event2	Required minimum $(\tau_2 - \tau_1)$	Computed $[d, D]$ $d \leq (\tau_2 - \tau_1) \leq D$	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	InpStable_M1	M1prech−	0	[0.7215, 2.2620]	None
	InpStable_A1	A1prech−	0	[0.9520, 1.4880]	None
	InpStable_A2	A2prech−	0	[2.4360, 3.5240]	None
	*InpStable_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	Event2	Required minimum $(\tau_2 - \tau_1)$	Computed $[d, D]$ $d \leq (\tau_2 - \tau_1) \leq D$	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	InpStable_M1	M1prech−	0	[0.1930, 3.2330]	None
	InpStable_A1	A1prech−	0	[0.6750, 1.7560]	None
	InpStable_A2	A2prech−	0	[1.8920, 4.0680]	None
	*InpStable_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

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

Table 5.3: Timing verification results for typical-case SPICE delays with $\pm 5\%$ (top table) and $\pm 10\%$ (bottom table) variations.

delays due to manufacturing and operating condition variations.

The timing verification experiments did not find any bugs in the final version of the DiffEq design. However, experiments were performed to determine how the correctness of the design depends on the delay of the A1M signal (an inter-controller synchronization signal) in Fig. 5.8b. This was motivated by feedback from the designers, who had observed by means of extensive SPICE simulations that in an earlier version of the design, where the delay of the buffer driving A1M was small, the circuit could potentially malfunction. To investigate this, the delay of the buffer driving the A1M signal was deliberately reduced in the timing constraint graph model and the analysis repeated. The computed bounds immediately indicated potential setup-time violations of domino circuits in the system. The entries marked with a * in Table 5.3 show some of these results.

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]

All times are in ns.

Table 5.4: Performance analysis results.

5.6 Summary

This chapter characterized tightly-coupled systems – a class of choice-free systems with bounded delays and min and max timing constraints – and described a polynomial-time algorithm for computing conservative bounds on the time separation of events in these systems. The algorithm is based on the same idea as Myers and Meng's algorithm [79]; nevertheless there are important extensions to and differences from their work. A complete asynchronous differential equation solver chip has been modeled and analyzed using the proposed algorithm, demonstrating its practical applicability.

There are several issues that still need to be addressed. The conditions in Definition 5.1 are sufficient to ensure monotonic convergence and finite bounds on the long-term time separation of events, even when we have no knowledge about the initial behavior of the system. However, it is likely that systems that are not strictly tightly-coupled in the sense of Definition 5.1 can also be meaningfully analyzed using the algorithm proposed in this chapter. A better characterization of such systems might provide greater insight into the types of systems that can be analyzed efficiently and with a fairly high degree of accuracy. It is also important that the current analysis be extended to systems with choice, so that it is applicable to a larger class of systems. A better characterization of the computed delay bounds is also needed. Specifically, it would be useful to be able to know whether the computed bounds are exact or potentially inexact, without having to compare them with the results of an expensive, exact algorithm.

Chapter 6

Conclusion

To design asynchronous circuits that are useful in practice, practical asynchronous CAD tools must exploit knowledge of the timing of their components and the environment. Hence, timing analyzers will be essential asynchronous design tools. The vast majority of existing asynchronous timing analyzers, however, perform exact analysis in worst case exponential time. While it has been shown in the literature that these algorithms run in reasonable time for small to medium sized system, their feasibility for analyzing large, real-world systems remains to be demonstrated. This thesis provides an alternative approach to the timing analysis problem. Specifically, it is shown that polynomial-time approximate algorithms for important timing analysis problems can be carefully designed so that they have a fairly high degree of accuracy in practice. As a result, polynomial-time algorithms hold a lot of promise for practical timing analysis of asynchronous systems in large, real-world applications.

This thesis does not recommend that exact algorithms with worst case exponential complexity be replaced by polynomial-time approximate algorithms. Instead, a more pragmatic approach is recommended, where polynomial-time approximate algorithms are used as a filter to weed out the vast majority of potential problems. Since polynomial-time algorithms have been shown to work well in practice in this thesis, a large system should first be analyzed with a polynomial-time algorithm to determine if there are any potential timing problems. Since the results computed by the algorithms presented in this thesis are provably conservative, any timing problem that actually exists in the circuit is guaranteed to be flagged by the approximate analysis. However, there might be false alarms as well – problems that are indicated by the approximate analysis, but which do not actually occur in practice. Once the set of potential problems has been narrowed down, more expensive but exact timing analysis techniques can be used to analyze the corresponding region of the system in greater detail. This can lead to significant savings in analysis time compared to the scenario where an exact algorithm is directly applied to detect all timing problems in the system.

6.1 Future Work

Although important problems in timing analysis of asynchronous systems have been addressed in this thesis, there are several issues that still need to be addressed.

- The min-max timing simulation algorithm in Chapter 2 computes conservative signal propagation delay bounds in the worst case. It would be interesting to characterize, if possible, the structure of the circuits for which the bounds are exact.
- The timing constraint graphs in Chapters 4 and 5 are currently generated manually. This is a very tedious and error prone task, especially when modeling large systems, such as the differential equation solver. Fully automated or partially automated interactive techniques for constructing timing constraint graphs from other system representations, such as an RTL model, is an important problem that needs to be addressed.
- The algorithms in Chapters 4 and 5 allow only one type of constraint (min or max) per event. Several researchers have considered the addition of linear constraints to systems with both min and max constraints. While the algorithm in Chapter 4 can be extended to handle systems with min, max and linear constraints by initializing the Δ matrix with the given bounds of linear constraints, the quality of results obtained by this simple extension has not yet been investigated. The problem of computing time separations in systems with max and linear constraints is particularly interesting because there is no known exact algorithm that is proven to run in polynomial-time, nor is there a proof of NP-completeness.

- Choice is an important feature of several practical systems. Unfortunately, the algorithms in Chapters 4 and 5 are not applicable to systems with choice. Existing algorithms for analyzing systems with choice have exponential time complexity. It would be interesting to investigate approximate algorithms for such systems that run in polynomial-time.

- The analysis techniques described in this thesis assumed that component delays are independent of each other. An important research problem is to extend the proposed techniques to take into account correlation between component delays.

- While the bounded delay model is useful for timing verification purposes, a more useful delay model for performance analysis is the *probabilistic delay model*. In this model, a probability distribution is specified for the delay of each component, and in the most general case, correlation between distributions of different delays are also specified. The problem of analyzing circuits with probabilistic delays has been addressed by some researchers [80, 86, 54, 37]. However, existing works either address restricted versions of the problem or use computationally expensive techniques like Monte-Carlo simulation. An investigation of more efficient techniques for probabilistic timing analysis of asynchronous circuits is an important and interesting research topic.

Bibliography

[1] *Hitachi High Speed CMOS Gate Array: HG62E Series Design Manual*.

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

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

[4] T. Amon, H. Hulgaard, S. Burns, and G. Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. In *Proceedings of the IEEE International Conference in Computer Design: VLSI in Computers and Processors*, pages 166–173, October 1993.

[5] Tod Amon. *Specification, Simulation and Verification of Timing Behavior*. PhD thesis, University of Washington at Seattle, 1993.

[6] P. Beerel and T. H.-Y. Meng. Automatic gate-level synthesis of speed-independent circuits. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 581–586, November 1992.

[7] M.R.C.M. Berkelaar. *lp_solve: Documentation and Release Notes*. Eindhoven Institute of Technology, Eindhoven, The Netherlands, 1992.

[8] G. Borriello and R. H. Katz. Synthesis and optimization of interface transducer logic. In *Proceedings of the 1987 IEEE/ACM International Conference on Computer Aided Design*, pages 274–277, November 1987.

[9] Gaetano Borriello. *A new interface specification methodology and its application to transducer synthesis*. PhD thesis, University of California at Berkeley, 1988.

[10] K. Bowden. Design goals and implementation techniques for time-based digital simulation and hazard detection. In *Proceedings of the 1982 International Test Conference*, pages 147–152, 1982.

[11] M. A. Breuer and L. Harrison. Procedures for eliminating static and dynamic hazards in test generation. *IEEE Transactions on Computers*, C-23:1069–1078, October 1974.

[12] M.A. Breuer and A.D. Friedman. *Diagnosis and Reliable Design of Digital Systems*. Computer Science Press, 1976.

[13] E. Brunvand and R. F. Sproull. Translating concurrent programs into delay-insensitive circuits. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 262–265, November 1989.

[14] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

[15] J. A. Brzozowski and J. C. Ebergen. Recent developments in the design of asynchronous circuits. Technical Report CS-89-18, University of Waterloo, Computer Science Department, 1989.

[16] J. A. Brzozowski, T. Gahlinger, and F. Mavaddat. Consistency and satisfiability of waveform timing specifications. *Networks*, January 1991.

[17] 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*, pages 152–155, November 1993.

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

[19] 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*, pages 190–194, November 1997.

[20] S. Chakraborty and D. L. Dill. More accurate polynomial-time min-max timing simulation. In *Proceedings of the Third International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 1997.

[21] 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*, pages 101–111, April 1997.

[22] S. Chakraborty, K. Y. Yun, and D. L. Dill. Practical timing analysis of asynchronous circuits using time separations of events. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 455–458, May 1998.

[23] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell. Delay fault models and test generation for random logic sequential circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 165–72, 1992.

[24] P. Chou and G. Borriello. Interval scheduling: Fine-grained code scheduling for embedded systems. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 462–467, June 1995.

[25] T.-A. Chu. *Synthesis of Self-timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT, 1987. Technical Report MIT-LCS-TR-393.

[26] W. A. Clark. Macromodular computer systems. In *Proceedings of the Spring Joint Computer Conference (AFIPS)*, volume 30, pages 335–336, April 1967.

[27] W. S. Coates, A. L. Davis, and K. S. Stevens. The Post Office experience: Designing a large asynchronous chip. *INTEGRATION, the VLSI Journal*, 15(4):341–366, 1993.

[28] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

[29] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[30] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Event suppression: Improving the efficiency of timing simulation for synchronous digital circuits. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 13(6):814–822, June 1994.

[31] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Verification of asynchronous interface circuits with bounded wire delays. *IEEE Journal of VLSI Signal Processing*, 7(1-2):161–182, February 1994.

[32] David L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. MIT Press, Cambridge, MA, 1989.

[33] D. Doukas and A. S. LaPaugh. Clover: A timing constraints verification system. In *Proceedings of the ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digitial Systems*, 1990.

[34] J. C. Ebergen. *Translating Programs into Delay-Insensitive Circuits*. Amsterdam: Centrum voor Wiskunde en Informatica, CWI Tract 56, 1989.

[35] J. C. Ebergen. A formal approach to designing delay-insensitive circuits. *Distributed Computing*, 5(3):107–119, 1991.

[36] E. B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM Journal of Research and Development*, 9(2):90–99, March 1965.

[37] M. A. Escalante and N. J. Dimopoulos. A probabilistic approach to timing analysis for synthesis and its application to microprocessor interface design. Technical report, Dept. of Electrical and Computer Engg., University of Victoria, BC CANADA, June 1994. Technical Report No. ECE-94-6.

[38] G. Fantauzzi. An algebraic model for the analysis of logical circuits. *IEEE Transactions on Computers*, C-23:576–581, June 1974.

[39] T. Gahlinger. *Coherence and Satisfiability of Waveform Timing Specifications*. PhD thesis, University of Waterloo, 1990.

[40] P. Girodias, E. Cerny, and W. J. Older. Solving linear, min and max constraint systems using CLP based on relational arithmetic. *Theoretical Computer Science*, 173(1):253–281, February 1997.

[41] J. Gunawardena. Min-max functions. *Discrete Event Dynamic Systems: Theory and Applications*, 4(4):377–407, September 1994.

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

[43] S. Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1):69–93, January 1995.

[44] D. A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, (3):161–190, March 1954.

[45] D. A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, (4):275–303, April 1954.

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

[47] 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*, pages 2–11, November 1994.

[48] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello. An algorithm for exact bounds on time separation of events in concurrent systems. Technical report, Dept. of Computer Sc. and Engg., Univ. of Washington, February 1994. Technical Report No. UW-CSE-94-02-02.

[49] 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*, 44(11):1306–1317, November 1995.

[50] N. Ishiura. *Studies on Logic Simulation and Hardware Description Languages*. PhD thesis, Kyoto University, Kyoto, Japan, 1990.

[51] N. Ishiura, Y. Deguchi, and S. Yajima. Coded time-symbolic simulation using shared binary decision diagram. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 130–145, 1990.

[52] N. Ishiura, M. Takahashi, and S. Yajima. Time symbolic simulation for accurate timing verification of asynchronous behavior of logic circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 497–502, 1989.

[53] F. Jin, E. Cerny, and H. Hulgaard. Maximum time separation of events in cyclic systems with linear and latest timing constraints. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design (to appear)*, November 1998.

[54] H.-F. Jyu, S. Malik, S. Devadas, and K. Keutzer. Statistical timing analysis of combinational logic circuits. *IEEE Transactions on VLSI Systems*, 1(2):126–137, June 1993.

[55] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *Proceedings of the 31st ACM/IEEE Design Automation Conference*, 1994.

[56] D. C. Ku and G. De Micheli. Relative scheduling under timing constraints: Algorithms for high level synthesis of digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(6):696–717, June 1992.

[57] 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, pages 221–225, October 1994.

[58] D. S. Kung. Hazard-non-increasing gate-level optimization algorithms. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 631–634, November 1992.

[59] W.K.C. Lam and R.K. Brayton. *Timed Boolean Functions: A Unified Formalism for Exact Timing Analysis*. Kluwer Academic Publishers, 1994.

[60] L. Lavagno, K. Keutzer, and A.L. Sangiovanni-Vincentelli. Synthesis of hazard-free asynchronous circuits with bounded wire delays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1):61–86, January 1995.

[61] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for Synthesis and Testing of Asynchronous Circuits*. Kluwer Academic Publishers, 1993.

[62] L. Lavagno and A.L. Sangiovanni-Vincentelli. Linear programming for optimum hazard elimination in asynchronous circuits. *Journal of VLSI Signal Processing*, 7(1):137–60, February 1994.

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

[64] M. H. Linderman. *Simulation of Digital Circuits in the Presence of Uncertainty*. PhD thesis, Cornell University, 1994.

[65] A. R. Martello. *Temporal Analysis for Time Bounded Causal Digital Systems*. PhD thesis, University of Pittsburgh, 1993.

[66] A. J. Martin. Programming in VLSI: From communicating processes to delay-insensitive VLSI circuits. In C. A. R. Hoare, editor, *UT Year of Programming Institute on Concurrent Programming*. Addison-Wesley, 1990.

[67] A. J. Martin, S. M. Burns, T.K. Lee, D. Borković, and P.J. Hazewindus. The design of an asynchronous microprocessor. In C. L. Seitz, editor, *Proceedings of the Decennial*

Caltech Conference on Very Large Scale Integration, pages 351–373. MIT Press, Cambridge, MA, 1989.

[68] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In W. J. Dally, editor, *Proceedings of the Sixth MIT Conference in Advanced Research in VLSI*, pages 263–278. MIT Press, Cambridge, MA, 1990.

[69] F. Mavaddat and T. Gahlinger. On deducing timing constraints in the verification of interfaces. *Formal Methods in System Design*, 12(3):223–239, April 1998.

[70] E. J. McCluskey. *Introduction to the Theory of Switching Circuits*. McGraw-Hull, New York, NY, 1965.

[71] E. J. McCluskey. *Logic Design Principles With Emphasis on Testable Semicustom Circuits*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[72] 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*, pages 48–51, October 1992.

[73] T. M. McWilliams. Verification of timing constraints on large digital systems. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 139–147, 1980.

[74] T. H.-Y. Meng. *Asynchronous design for digital signal processing architectures*. PhD thesis, University of California, Berkeley, November 1988.

[75] R. E. Miller. *Switching Theory, Volume II: Sequential Circuits and Machines*. John Wiley and Sons, New York, 1965.

[76] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 52–57. Computer Science Press, Inc., 1990.

[77] C. E. Molnar, T.-P. Fang, and F. U. Rosenberger. Synthesis of delay-insensitive modules. In Henry Fuchs, editor, *1985 Chapel Hill Conference on Very Large Scale Integration*, pages 67–86. Computer Science Press, Inc., 1985.

[78] C. J. Myers. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*. PhD thesis, Dept. of Elec. Eng., Stanford University, 1995.

[79] C. J. Myers and T. H.-Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, 1(2):106–119, June 1993.

[80] A. Nadas. Probabilistic PERT. *IBM Journal of Research and Development*, 23(3):339–347, May 1979.

[81] C. D. Nielsen and M. Kishinevsky. Performance analysis based on timing simulation. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 70–76, October 1994.

[82] S. M. Nowick. *Automatic Synthesis of Burst-mode Asynchronous Controllers*. PhD thesis, Stanford University, 1993.

[83] S. M. Nowick, K. Y. Yun, and D. L. Dill. Practical asynchronous controller design. In *Proceedings of the 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 341–345. IEEE Computer Society Press, October 1992.

[84] H. Schols. *Delay-insensitive Communication*. PhD thesis, Technische Universiteit Eindhoven, 1992.

[85] 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*, November 1995.

[86] D. R. Tryon, F. M. Armstrong, and M. R. Reiter. Statistical failure analysis of system timing. *IBM Journal of Research and Development*, 28(4):340–355, July 1984.

[87] J. Udding. *Classification and Composition of Delay-Insensitive Circuits*. PhD thesis, Technische Universiteit Eindhoven, 1984.

[88] E. Ulrich, K.P. Lentz, S. Demba, and R. Razdan. Concurrent min-max simulation. In *EDAC. Proceedings of the European Conference on Design Automation*, pages 554–557, 1991.

[89] S. H. Unger. *Asynchronous Sequential Switching Circuits*. Wiley-Interscience, New York, NY, 1969.

[90] P. Vanbekbergen. *Synthesis of asynchronous controllers from graph-theoretic specifications*. PhD thesis, Interuniversitair Micro-Elektronica Centrum, 1993.

[91] 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*, pages 302–306, March 1992.

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

[93] E. A. Walkup and G. Borriello. Interface timing verification with application to synthesis. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 106–112, June 1994.

[94] T. E. Williams. Performance of iterative computation in self-timed rings. *Journal of VLSI Signal Processing*, 7(1-2):17–31, February 1994.

[95] G. Winskel. An introduction to event structures. *Linear Time, Branching Time and Partial Orders in Logics and Models for Concurrency*, pages 364–397, 1989.

[96] 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*, pages 64–75, April 1997.

[97] T.-Y. Yen, A. Ishii, A. Casavant, and W. Wolf. Efficient algorithms for interface timing verification. *Formal Methods in System Design*, 12(3):241–265, April 1998.

[98] T.-Y. Yen and W. Wolf. Performance estimation of real-time distributed embedded systems. In *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 64–69, October 1995.

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

[100] 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*, pages 140–153, April 1997.

[101] 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. *to appear in IEEE Transactions on VLSI Systems*, 6(3), September 1998.

[102] K. Y. Yun and D. L. Dill. Automatic synthesis of 3D asynchronous finite-state machines. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer Aided Design*, pages 576–580. IEEE Computer Society Press, November 1992.

[103] K. Y. Yun and D. L. Dill. Unifying synchronous/asynchronous state machine synthesis. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer Aided Design*, pages 255–260. IEEE Computer Society Press, November 1993.

[104] K. Y. Yun, D. L. Dill, and S. M. Nowick. Synthesis of 3D asynchronous state machines. In *Proceedings of the 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 346–350. IEEE Computer Society Press, October 1992.

[105] K. Y. Yun, D. L. Dill, and S. M. Nowick. Practical generalizations of asynchronous state machines. In *Proceedings of The European Conference on Design Automation with The European Event in ASIC Design*, pages 525–530. IEEE Computer Society Press, February 1993.