Interface Timing Verification with Application to Synthesis

Elizabeth A. Walkup*, Gaetano Borriello†

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

Abstract – A fundamental timing analysis problem in the verification and synthesis of interface logic circuitry is the determination of allowable time separations, or *skews* between interface events, given timing constraints and circuit propagation delays. These skews are used to verify timing properties and determine allowable propagation delays for logic synthesis. This paper presents an algorithm that provides tighter skew bounds with better asymptotic running time than previous methods, and shows how to apply the method to synthesis tasks.

I Introduction

Temporal behavior of interface circuitry is frequently described using event-based representations that relate the occurrence times of events with timing constraints and propagation delays [1, 2, 3, 4, 5, 6]. In this paper, we present an efficient solution to a key problem in the verification and synthesis of interface *glue logic*, namely, the determination of tight bounds on the temporal separations between events. To verify a synthesized circuit, we must be able to check that the circuit's outputs will occur within the time interval required and expected by the circuit's environment. In synthesizing the circuit, we must be able to determine the amount of delay within which the logic may generate an interface event. This permits optimizing the logic to take advantage of the temporal characteristics of the interface. The basic subproblems of both these tasks can be phrased in terms of bounds on the skew between pairs of events.

Previous work on this problem has suffered from a combination of two deficiencies. First, existing verification algorithms are inefficient. The method in [1] relies on exponential search, while the method of [3] does not produce the tightest possible skew bounds and has a running time which depends intimately upon the time bounds of the constraints. Second, they have not been useful for the synthesis process because they yield very loose bounds in the presence of unknown delays, a common situation before a circuit is synthesized.

In this paper, we first present an interface timing specification model that unifies the concepts of timing constraint and propagation delay into a single constraint type. We then provide an efficient algorithm for solving systems of these constraints. The algorithm yields tight bounds even in the presence of unknown constraint bounds, and its worst case running time can be expressed independently of the initial constraint values. We conclude with a discussion of how the algorithm can be used in both verification and synthesis applications.

II Interface Timing Specification

Interface specifications consist of a sequence of events, which are transitions on signal wires. Such a specification can be viewed as a partial ordering of the events and the ways in which they can be spaced in time. Temporal relationships between these interface events are expressed with *propagation delays* and *timing constraints*. In this section, we explain the semantic difference between these two types of temporal constraints and present a model that expresses both of them in a unified form.

A An Interface Specification Example

Suppose we wish to synthesize a circuit to interface with an SRAM. We say that the SRAM is then the *environment* for our interface circuit. Figures 1 and 2 provide the interface specification for a simplified SRAM read operation – any circuit we synthesize to interface with the

*Supported in part by an NSF Graduate Fellowship.

†Supported by PYI Award (MIP-8858782) and by the DARPA/CSTO Microsystems Program under an ONR monitored contract (N00014-91-J-4041).

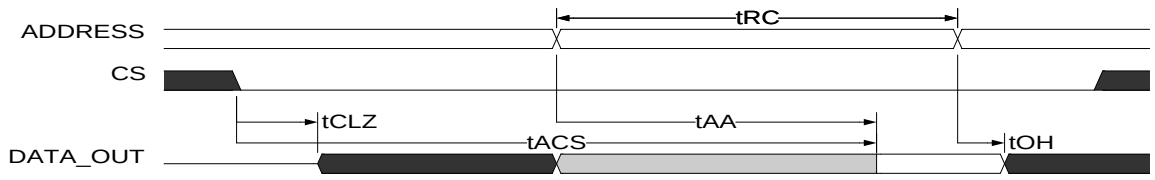

Fig. 1: Timing diagram for an SRAM read operation.

Propagation Delay Values for SRAM				
name	from	to	min	max
t_{AA}	Address Valid	Data Valid	0	20
t_{ACS}	CS low	Data Valid	0	20
t_{CLZ}	CS Low	Data Driven	5	
t_{OH}	Adress Invalid	Data Invalid	5	
Performance Requirement for SRAM				
t_{RC}	Address Valid	next Address	100	

Fig. 2: Constraint values for the SRAM example.

SRAM must adhere to the performance requirements in Figure 2, and may take advantage of the propagation delay information to meet any further timing constraints on its own performance. In this example, the appearance of valid data on the $DATA\_OUT$ line is the result of a propagation delays from both the lowering of the signal CS and the assertion of a valid address on the *Address* lines. Throughout the remainder of this paper, these three events will be referred to as DV,CS, and AV, respectively.

Propagation delays, or *delay constraints* such as these express structural dependencies between the inputs and outputs of both the interface circuitry and the environment. These constraints, here expressed as ranges of 0 to 20 time units from the lowering of CS and the appearance of a valid address, determine when valid data will first appear. The data appears at the maximum of $CS+t_{ACS}$ and $AV + t_{AA}$ where t_{ACS} and t_{AA} are within the 0 to 20 time unit delays listed for DV relative to AV and CS. Note that this event may actually occur outside the range specified by either input event's propagation delay taken alone. Therefore, we consider these constraints linked or *dependent* on one another. We can express these as:

$$Max\left\{\begin{array}{c} x_{j_1}+\delta_{j_1,i}, \\ \vdots \\ x_{j_m}+\delta_{j_m,i} \end{array}\right\} \leq x_i \leq Max\left\{\begin{array}{c} x_{j_1}+\Delta_{j_1,i}, \\ \vdots \\ x_{j_m}+\Delta_{j_m,i} \end{array}\right\},$$

where δ and Δ represent the lower and upper bounds of the propagation delays and the x_i's are individual events. With propagation delays, the Max term causes an event to happen only after all predecessor events plus their corresponding delay have occurred.

The other constraint type, which we term *timing constraints*, come in two flavors: *requirements*, which the environment imposes upon the circuit for proper interaction, and *guarantees*, which describe the operating environment independently of the underlying implementation. An example of the first type would be the minimum time constraint t_{RC} on how long the address must remain valid. An example of the second would be an environment asserting that it will never change two signal values within a short interval of each other. Constraints of this type are *independent* of one another and specify the exact time range within which one event must occur relative to another. Performance requirements of the circuit can also be viewed as timing constraints – specifying that an output response must be seen within a particular interval. We can express these as:

$$x_j + \delta_{j,i} \leq x_i \leq x_j + \Delta_{j,i},$$

where δ and Δ represent the lower and upper bounds of the constraint.

Previous work has used different models for temporal constraints that make more explicit distinctions between the two types of constraints. McMillan and Dill ([3]) use the terms Linear and Max constraints for timing and delay constraints, respectively. Vanbekbergen ([4]) has a more complete yet, not largely useful, taxonomy that labels timing and delay constraints as *type 1* and *type 2*, respectively. We find it more useful to translate both types into inequalities involving the Max operation. We can express both types of constraints as a system of inequalities of the following form:

$$x_i \leq Max\{x_{j_1} + \Delta_{j_1,i}, \ldots, x_{j_m} + \Delta_{j_m,i}\}. \qquad (1)$$

Since timing constraints are independent, there is only one term in the Max expression – reducing Equation 1 to a simple arithmetic inequality.

Suppose that we are given an interface circuit for the SRAM of Figures 1 and 2 which meets the performance guarantees of Figure 3. The set of equations describing the relative times of events AV, DV, and CS are:

Performance Guarantees for SRAM Interface			
from	to	min	max
Address Valid	CS low		300
CS low	Data Valid	30	

Fig. 3: Performance bounds for an SRAM interface.

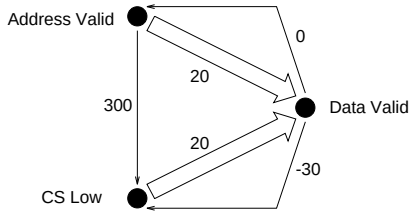

Fig. 4: Graphical representations of constraints in the SRAM interface. Outline arcs represent interdependent propagation delays; thin arcs represent independent constraints.

$$\begin{aligned} DV &\leq Max(AV+20, CS+20) \\ AV &\leq Max(DV+0) \\ CS &\leq Max(DV-30) \\ CS &\leq Max(AV+300)\ . \end{aligned}$$

Systems of these of events can be abstracted as a constraint graph over interface events. We say a given set of constraints *induces* a graph whose nodes represent the events, and whose arcs, from x_j to x_i with label δ represent each of the terms $x_j + \delta$ in a constraint with x_i on the left hand side of the inequality. The graph induced by the set of constraints given above is shown in Figure 4.

III The Verification Problem

We can verify that a system's *required* performance constraints are met by determining that the *maximum skew* between all interface and environment events in the system meet all *performance requirements* of the system.

A Formal Problem Definition

We now state the verification problem more formally. Given

- $\mathcal{X} = \{x_0, x_1, \ldots, x_{n-1}\}$ a set of occurrence times of events in the system
- $\mathcal{C}$, a set of constraints c_j of the form:

$$c_j : x_i \leq Max\{x_{j_1} + \delta_{j_1,i}, \ldots, x_{j_m} + \delta_{j_m,i}\},$$

determine either a tight upper bound on the occurrence times of all variables $x_1, \ldots, x_{n-1}$ relative to $x_0 = 0$, or that the set of inequalities is inconsistent.

In practical applications, one would apply the verification algorithm to a fully synthesized combined circuit-environment specification with all performance *requirements* removed and then check that the bounds given by the verification algorithm are no looser than any performance *requirement*. We do not remove propagation delays and performance guarantees since they determine how the circuit and its environment will react.

B Previous Work

Algorithms for determining the maximum inter-event timing separations have been proposed by Borriello [1] and McMillan and Dill [3]. The algorithm of [1] is exponential in the number of nodes with *propagation delays* and can quickly become too costly for large composed graphs. The implementation is straightforward and uses backtracking to determine which causal relationships determine the occurrence time of an event.

The algorithm given in [3] has two drawbacks: in many practically interesting cases, it provides infinite separation bounds between events with finite bounds; and its worst case running time depends not only upon n, the number of events in the system, but also upon the $\delta_{i,j}$'s, the bounds of the constraints. In this algorithm, initial infinite upper bounds on node separations are refined by successive applications of appropriate constraints from the input set. The problem with this approach, as noted in [3], is that the running time of the algorithm can depend on the values of the constraints, giving a worst case complexity of $O(n^3 \cdot \sum |\delta_{i,j}|)$. This behavior occurs precisely when there is a "negative cycle" in the graph with at least one arc of the cycle belonging to a *propagation delay*. When applied to the SRAM example of Figure 4, the number of times the algorithm of [3] (hereafter referred to as the $\mathcal{MD}$ algorithm) applies the constraints $DV \leq Max(AV+20, CS+20)$ and $CS \leq DV - 30$ is dependent upon on the value of the 300 ns constraint from AV to CS. Increase the 300ns constraint to 600ns and the algorithm takes twice as long.

In addition, the limit of CS's maximum skew relative to AV as the 300 ns constraint is raised towards infinity is -10, indicating that the constraint is redundant. However, if the constraint is completely removed, the algorithm will give a final bound of ∞ for CS relative to AV. If we assume that all events must occur eventually then an infinite bound simply indicates that we do not know the relationship between event occurrence times. In this case, an infinite maximum skew between the events is wrong: we know that they will occur and that CS must occur at least 10 ns before AV.

C An Improved Verification Algorithm

We now introduce our new "short circuiting" verifica-

Optimized Constraint Relaxation Algorithm
Input: Event set $\mathcal{X}$ and constraint set $\mathcal{C}$ **Result:** x_j contains tight upper bound on $(x_j - x_0)$

Set all bounds $x_j j \neq 0$ to symbolic quantity $\mathcal{V}$.
Set x_0 to 0.
Repeat:
  Repeat n **times:**
  "Update" subroutine:
  If a constraint c_i exists that can reduce the bound on an x_j,
  update x_j to reflect c_i and record c_i as the most recent to update x_j.
  Choose topologically first components of size ≥ 2 in the graph induced by such recorded constraints.
  Within each such component do (Short-circuit step):
  For all x_j whose recorded constraints contain arcs from exterior the component find
  (x_j's current value) - (the MAX value contributed from outside constraint arcs only).
  Let Δ be the smallest such difference in the component. (It will be positive.)
  Subtract Δ from all bounds x_j in the component.
Until $x_0 < 0$ or no x_j changes.

Fig. 5: $\mathcal{SC}$ constraint relaxation algorithm.

tion algorithm, hereafter referred to as the $\mathcal{SC}$ algorithm. It's improvements over the $\mathcal{MD}$ algorithm rely on two observations:

- If a "negative cycle" can be discovered, we can then predict how many times the constraints along that cycle can be re-applied. This information can be used to speed up the performance of the McMillan and Dill algorithm.
- Since we assume that all events will eventually happen, it is correct to define the problem using the limit of the maximum skews as an initial bound on all maximum skews goes to infinity. This allows us to accurately handle cases such as that of Figure 4 with the redundant 300 ns constraint removed.

If we define the *dependency graph* of the system to be the subgraph induced by those constraints which were used to provide the current bound on each node, then patterns of repeated constraint application appear as *strongly connected components* in this dependency graph. To calculate the limit of the maximum skews as an initial bound goes to infinity, we begin the algorithm by setting the maximum skews of all nodes in the graph to the symbolic constant $\mathcal{V}$, with the exception of one node whose time is set to 0 to serve as the origin of the time measurement. We assume that $\mathcal{V}$ is a very large number, and so perform all calculations involving it symbolically. An intuitive description of the algorithm follows; pseudocode is given in Figure 5.

The short circuit algorithm cycles through the following four steps:

- Pass through n rounds of the **Update** subroutine, where n is the number of events in the system. The **Update** subroutine applies to each event the constraint that most greatly reduces its bound. During this process, the dependency graph summarizing which constraint was used most recently to update each event's maximum skew is maintained. After n rounds, any current cyclic behavior will appear since every cycle has at most n nodes on it.
- Perform a strongly connected components analysis of the dependency graph. "Negative cycles" will appear as strongly connected components in this directed graph and may be any strongly connected component, not just simple cycles.
- Among such components of size ≥ 2, find the topologically first ones. These indicate the constraint dependencies which may be profitably "short circuited".
- For each of these components, find all constraints whose arcs have their tails outside the component (called *exterior arcs*) In Figure 6, the only such exterior arc is from AV to DV. When the constraint relaxation procedure is exhibiting cyclic behavior, it

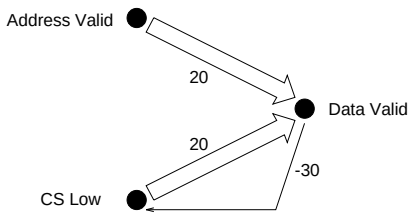

Fig. 6: Constraint set on which topological information is performed.

Convergence with Short-Circuit Algorithm				
	Number of Pass Through Outer Repeat Loop			
Node	start	1^{st} Updates	1^{st} SC	2^{nd} Updates
AV	0	0	0	0
CS	$\mathcal{V}$	$\mathcal{V}-50$	$\mathcal{V}-50$	-10
DV	$\mathcal{V}$	$\mathcal{V}-30$	20	20

Fig. 7: Applying the short circuit algorithm to the graph in Figure 4 with the redundant 300ns constraint removed.

will continue to do so until one of the exterior arcs provides the actual bound on the node it points to. We discover which node will limit the cycle by comparing the current skew bounds of all nodes that such exterior arcs enter with the value they would have if the *interior arcs* (those arcs with tails inside the component) were to be removed. Whichever of these has the least difference between the current and exterior-provided skew bounds is chosen as the "winner", and we update that node's skew to match the incoming arc.

Note that the last step is where the symbolic value $\mathcal{V}$ becomes useful – a component may have all nodes with values containing a $\mathcal{V}$ term when all exterior arcs provide potential bounds not containing $\mathcal{V}$. In such a case, the $\mathcal{MD}$ algorithm will erroneously calculate an infinite maximum skew for all nodes in the connected component. We assume that any value containing an $\mathcal{V}$ is larger than any value not containing $\mathcal{V}$, and this allows us to shortcircuit these components as well. Note that the use of $\mathcal{V}$ also allows us to apply *ShortCircuit* to systems that contain variables with true upper bounds of infinity. These variables will be precisely those whose final bound as given by the algorithm still includes a $\mathcal{V}$ term.

In Figure 7 we show the results of applying the short-circuit algorithm to the graph in Figure 4, without the redundant constraint $CS \leq AV - 300$ since we can now handle an initial upper bound of infinity on $CS - AV$.

D Practical Results

Each of the n update rounds takes time at most $|\mathcal{C}|$ where $|\mathcal{C}|$ is the number of terms $x_j + \delta_{j,i}$ in the constraint set. The topological information takes time at most $O(|\mathcal{C}|)$ to calculate. We have unfortunately been unable to determine a tight bound on the number of short circuiting passes that must be made in the worst case. It is our intuition, however that the number of required passes is polynomial and we have been unable to generate any example that takes more than $P = O(|\mathcal{C}|)$ such passes. The algorithm must be run once for each possible assignment of x_0, thus giving a bound of

$$n \cdot P \cdot (n \cdot |\mathcal{C}| + |\mathcal{C}|)$$

to determine all n^2 maximum event separations in the worst case, which we feel is probably n^6. For practical problems, $\mathcal{C}$ is $O(n)$, giving a likely bound of n^4. In contrast, the bound for the $\mathcal{MD}$ algorithm is $n^3 \cdot \sum |\delta|$ in the worst case and $n^2 \cdot \sum |\delta|$ for the practical case. We would expect that n^2 is much less than the sum of the δ's for practical problems. An absolute worst case on the number of passes required by our algorithm is $\mathcal{T}$, where $\mathcal{T}$ is the number of distinct rooted trees over the constraint set $\mathcal{C}$. This bounds the number of different dependency graphs we will see during the short circuiting portion of the algorithm – it can be shown that with each pass, the portions of the dependency graph which topologically precede all strongly connected components of size greater than one must be distinct.

We have implemented the algorithm and run both practical examples [7, 3] and randomly generated larger examples built to look like practical examples (i.e. similar constraint sizes and constraint type ratio). In these cases no more than three short circuiting phases were required to find maximum skews relative to a single event. Running times were on the order of 20 seconds on a DEC station 5000 to find all n^2 maximum skews for a dense constraint graph with 80 nodes, which is much larger than we expect to see in practice.

IV Applications to Synthesis

In this section, we define a *synthesis* problem over systems of propagation delays and performance constraints, and show why previous verifications algorithms are inadequate to perform synthesis tasks. We use an example problem to give an intuition for the differences between the verification and synthesis problems, and provide a new constraint taxonomy to distinguish the two problems.

A A Taxonomy of Constraint Types

Given a collection of devices whose temporal behavior is fully specified and a fully synthesized interface circuit, the verification problem determines that the interface circuit meets the timing requirements of the components it interconnects. In contrast, as we synthesize interface logic, we would like all the timing constraints provided to guide our synthesis process. In particular, we wish to utilize the circuit's required timing constraints to determine allowable propagation delays for that circuit. To accomplish this, we partition the types of delays encountered during the synthesis procedure into the following two orthogonal categories:

Specification Semantics	Constrainable	Unconstrainable
Propagation Delays:	*incompletely synthesized logic*	*fully synthesized logic*
Guarantees:	*modifyable environment*	*behavior of environment*
Requirements:	*all performance constraints*	*do not exist*

Fig. 8: Taxonomy of interface constraint types.

Propagation Delays vs. Timing Constraints: Both circuit and environment may include structural timing information in the form of propagation delays. The environment may include timing requirements which indicate the allowable time separations of inputs to the environment and timing guarantees which summarize its temporal behavior.

Constrainable vs. Unconstrainable ranges: Constrainable delays and performance measures indicate time ranges which may be further constricted, as needed, to create a consistent circuit-environment combination; unconstrainable ranges represent elements for which the circuit must function correctly for arbitrary delay behavior anywhere within the given range.

Figure 8 gives a summary of these categories, which are here more completely described from top to bottom.

- **Constrainable Propagation Delays:** Timing behavior for which logic is either not completely synthesized, in the case of interface circuitry, or for which timing behavior can be modified, in the case of the environment.
- **Unconstrainable Propagation Delays:** Timing behavior for which logic is already synthesized.
- **Constrainable Timing Guarantees:** Environment timing behavior that is modifyable, but for which no explicit structural information is provided.
- **Unconstrainable Timing Guarantees:** Unmodifyable environment behavior for which no explicit structural information is provided.
- **Constrainable Timing Requirements:** Performance requirements may always be over-met.
- **Unconstrainable Timing Requirements:** These cannot exist.

Under this taxonomy, the verification problem consists of checking that a fully specified system (no portions still represented with constrainable propagation delays or constrainable guarantees) meets all of its performance requirements, and the synthesis problem consists of constraining all constrainable propagation delays and constrainable guarantees until they, in conjunction with their unconstrainable counterparts meet all of the timing requirements.

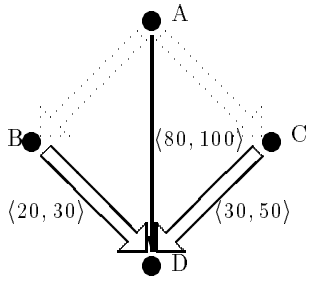

Fig. 9: A simple synthesis problem. Dotted arcs need to be synthesized; outline arcs represent propagation delay; thin lines represent timing requirements. For visual clarity, constraints are in double-bounded form with $\langle \delta, \Delta \rangle$ indicating the lower and upper bounds on the constraint.

The algorithms of [1] and [3] are both *verification algorithms*, meaning that given a set of constraints, they determine maximum bounds on the separation of signal events given that all constraints hold. In contrast, a *synthesis algorithm* must determine bounds on the time available for a circuit to generate an event to ensure that the desired constraints hold.

B Solutions are Inherently Disjoint

Consider the simple system in Figure 9. The environment provides output A, and some time later expects two inputs, at B and C, from which it generates the output at D. If we must synthesize the logic that produces B and C, we have two problems:

- They must be synthesized quickly enough after A occurs that the environment can produce D within the 100 ns maximum time from A.
- At least one of them must produce output late enough to keep D from happening before it's minimum 80 ns time bound from A.

Note that we need not cause both B and C to occur later in order to meet the minimum time constraint. If we synthesize logic to produce B from A within the delay range $\langle 30, 60\rangle$ then we must synthesize C from A with delay exactly 50. Alternatively, if we generate C from A with delay range $\langle 20, 40\rangle$, we can synthesize B from A anywhere in the range $\langle 60, 70\rangle$. Note that these are two disjoint solutions.

C Towards a Synthesis Algorithm

The differences between the verification and synthesis problems are fundamental – for a given system there is one correct answer to the verification problem, but perhaps none or many correct synthesis solutions. However, we note that the following steps can greatly improve the bounds returned by the verification algorithm.

- Perform the verification algorithm *only* on the UNCONSTRAINABLE constraints to get *unconstrainable skews*.
- Perform the verification algorithm *only* on the CONSTRAINABLE constraints to get *constrainable skews*.
- Replace each *unconstrainable skew* of the form $x_i - x_j \leq \delta$ with a constraint $x_i - x_j \geq \delta$.
- Create new *synthesis constraints* as possible from combinations of one *constrainable skew* and one of the "flipped" *unconstrainable skews*
- Run the algorithm on this last set of *synthesis constraints*.

The resulting bounds on skews between nodes connected by CONSTRAINABLE DELAY constraints provide tighter bounds on how those arcs may be synthesized than are obtainable from the verification algorithm alone. Bounds on CONSTRAINABLE relationships are narrowed subject to UNCONSTRAINABLE constraints taking on their worst case delays. This procedure essentially requires that all TIMING REQUIREMENTS hold no matter where an UNCONSTRAINABLE DELAY occurs within its allowed range. The bounds can then be used to guide an iterative heuristic synthesis procedure to determine final bounds on all CONSTRAINABLE DELAYS and CONSTRAINABLE GUARANTEES.

V Conclusions and Future Work

We have discussed the differences in verifying and synthesizing interface circuit systems specified with propagation delays, performance guarantees, and performance requirements, and provided a taxonomy of constraint types to express the range of desired behaviors. A new algorithm was presented for satisfying systems of constraints as arise in interface timing verification. This algorithm improves upon the previous work of McMillan and Dill [3] in two ways: it robustly handles infinite delay bounds, and its worst case running time is not dependent on the individual delay values of the constraints. We have proven the algorithm correct (the complete algorithm and proof of correctness can be found in [8]) , and shown that it is practically applicable. In addition, we have shown how to modify the verification algorithm to more readily handle synthesis tasks.

Currently we are working on determining the verification algorithm's theoretical time performance bounds, as well as developing a full synthesis procedure.

Acknowledgments

The authors wish to thank David Dill, Peter Vanbekbergen, Martin Tompa, and Paul Beame for many useful discussions of material contained herein.

References

[1] Gaetano Borriello. *A New Interface Specification Methodology and its Application to Transducer Synthesis.* PhD thesis, University of California, May 1988. Report No. UCB/CSD 88/430.

[2] Thomas Gahlinger. *Coherence and Satisfyability of Waveform Timing Specifications.* PhD thesis, University of Waterloo, 1990. Research Report CS-90-11.

[3] Kenneth Mc Millan and David Dill. Algorithms for interface timing verification. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1992.

[4] Peter Vanbekbergen, Gert Goossens, and Hugo De Man. Specification and analysis of timing constraints in signal transition graphs. In *Proceedings of the European Design Automation Conference*, March 1992.

[5] Peter Vanbekbergen. *Synthesis of Asynchronous Controllers from Graph-Theoretic Specifications.* PhD thesis, Katholieke Universiteit Leuven, September 1993.

[6] Bruce Gladstone. Specification of timing in a digital system. *ASIC and EDA*, pages 46–52, August 1993.

[7] Chris Myers. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, June 1993.

[8] Elizabeth A. Walkup and Gaetano Borriello. Interface timing verification with combined max and linear constraints. Technical Report 94-03-04, University of Washington Department of Computer Science, March 1994.