Performance Analysis Based on Timing Simulation

Nielsen, Christian Dalsgaard; Kishinevsky, Michael

Published in:
A C M / I E E E Design Automation Conference. Proceedings

Publication date:
1994

Document Version
Publisher's PDF, also known as Version of record

Performance Analysis based on Timing Simulation

Christian D. Nielsen Michael Kishinevsky
Department of Computer Science
Technical University of Denmark
Building 344, Lyngby, DK-2800 [*]

Abstract — Determining the *cycle time* and a *critical cycle* is a fundamental problem in the analysis of concurrent systems. We solve this problem using *timing simulation* of an underlying Signal Graph (an extension of Marked Graphs). For a Signal Graph with *n* vertices and *m* arcs our algorithm has the polynomial time complexity *O(b^2m)*, where *b* is the number of vertices with initially marked in-arcs (typically *b*≪*n*). The algorithm has a clear semantic and a low descriptive complexity. We illustrate the use of the algorithm by applying it to performance analysis of asynchronous circuits.

I INTRODUCTION

Determining the *cycle time* and a *critical path* is a fundamental problem in the analysis of concurrent systems. The cycle time of a system is defined as the *average* time separation between equivalent *events* in the system. The sequence of events determining the cycle time, called the critical cycle, may be viewed as the bottleneck of the system.

We use a Signal Graph model with arcs labelled with real numbers to represent the problem. These numbers can be interpreted as delays between events and for this reason we call this model Timed Signal Graphs. The Signal Graph model is an extension of Marked Graphs[1] which includes means for modelling the initial behavior of systems, not only a totally cyclic behavior. Regardless of this, the algorithm we are presenting is just as applicable to Marked Graphs and to any other equivalent model, for example to event rules systems [2].

In [2], the problem of finding the cycle time of a circuit was reduced to a problem of linear programming with polynomial complexity. In [7], it is observed that the timing behavior of a Signal Graph (and similar models) can be studied as an eventually periodic behavior of the corresponding max-functions. The problem tackled in this paper can also be formulated as the problem of a parametric shortest path [13], or can be reduced to the minimum mean-weight cycle or minimum cost to time ratio cycles problems [1, 8, 11]. For a graph with n vertices and m arcs the best known algorithms run in $O(nm + n^2\ log\ n)$ time [13] or $O(T(m + n \cdot log\ n))$ [8], where parameter T is an upper bound on the length of each path and cycle in a graph.

This paper describes an algorithm that takes a Timed Signal Graph as input and returns as a result the cycle time and the critical cycle of the graph. Our algorithm runs in $O(b^2 m)$ time, where b is the number of vertices with initially marked in-arcs. In the worst case when all vertices have initially marked in-arcs it gives $O(n^2 m)$. However, typically $b \ll n$ and the algorithm demonstrates linear run-time. This algorithm is based on a simple calculation process called *timing simulation* and has a clear semantic and a low descriptive complexity.

We have applied our algorithm to the performance analysis of asynchronous circuits. The Signal Graph representation (and other similar models) may be viewed as a circuit specification as in [4, 9, 10, 12], or as an intermediate representation derived from other descriptional models. For the performance analysis of a circuit represented as a net-list, we apply the algorithm described in [9] to extract the Signal Graph.

The remainder of this paper is organized as follows. In Section II, we give an informal introduction to the algorithm. In Section III, we summarize the Signal Graph model. In Section IV, we develop the notion of timing simulation and derive how the cycle time may be calculated from an infinite timing simulation. In order to limit the timing simulation to finite length, we explore the correspondence between the cycle time and cycles in the Signal Graph (Section V). In Section VI, we utilize this knowledge to obtain the cycle time from timing simulations in finite time. In Section VII, we summarize the algorithm. Finally, Section VIII demonstrates the application of the algorithm to the performance analysis of asynchronous circuits and shows some illustrative examples. For the sake of brevity all propositions in this paper are given without proofs. They can be found in [3].

II INFORMAL PROBLEM DESCRIPTION

Let us illustrate the problem of finding the cycle time informally using the circuit shown in Figure 1a. It consists of a C-element, two NOR-gates and a buffer and has five signals: one for each gate output (a, b, c, f) and one for the input node of the circuit (e). To all the gate inputs, we have assigned a fixed propagation delay from this input to the output of the gate.

The graph in Figure 1b, called a *Timed Signal Graph*, represents the causal dependencies between the signal transitions of the circuit. An arrow which is crossed over (with a $\times$) is active once only, while the dots (●) indicate the particular initialization of the circuit. All arcs are labelled with the corresponding gate delays.

The cycle time of a circuit is determined by the longest simple cycle, called the *critical cycle*, in the corresponding graph representation [2]. For example, the critical cycle for the graph in Figure 1b is: $a\uparrow \xrightarrow{3} c\uparrow \xrightarrow{2} a\downarrow \xrightarrow{3} c\downarrow \xrightarrow{2} a\uparrow$. The length of this cycle is 10.

A straightforward approach for finding the critical cycle and, hence, the cycle time is to search for all cycles and to choose the longest. Unfortunately, the number of cycles may be exponential in the number of arcs in the graph.

A *timing diagram* for our example, shown in Figure 1.c, can be obtained from the Signal Graph by a calculation process called *timing simulation*. The dashed arrows in the timing diagram corresponds to the arcs of the graph. For the acyclic graphs timing simulation is analogous to the PERT-analysis [6].

The signals e and f change only once during the circuit

[*] This work has been supported by The Danish Technical Research Council.

[1] Marked Graphs are a subclass of Petri Nets [5]. In such nets every place possesses exactly one in- and one out-event, and no conflict situations are possible.

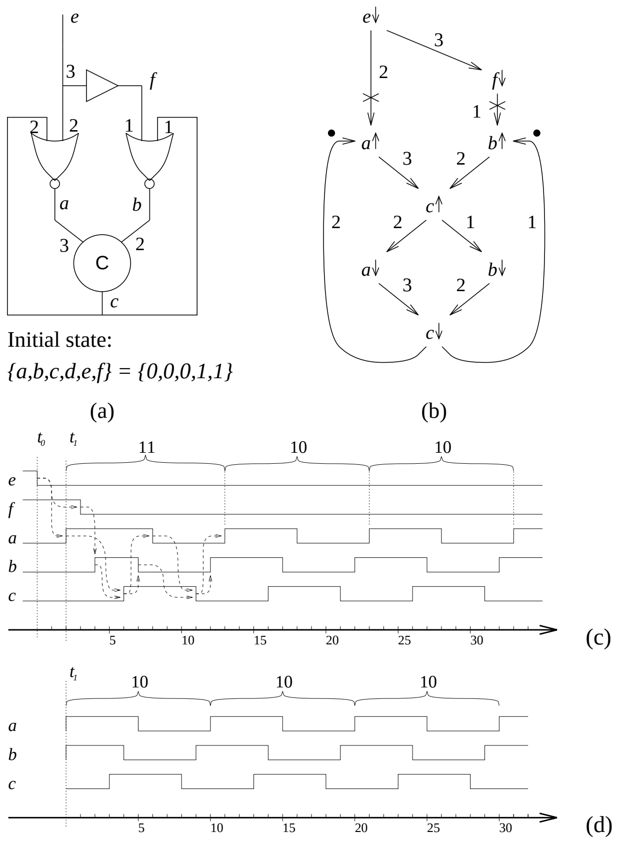

Figure 1: A circuit (a), its Timed Signal Graph (b), timing diagram (c) and $a{\uparrow}$-initiated timing diagram (d).

operation. The others, a, b and c oscillate infinitely. We call the up- and down-going transitions of such oscillating signals *repetitive events*.

The time separation between two successive identical events of a signal is called the *occurrence distance* between the events. The occurrence distance between, say, the first and the second up-going transitions of signal a ($a{\uparrow}^0$ and $a{\uparrow}^1$) is 11. After an initial period the oscillations on the three repetitive signals a, b, c stabilize with a fixed occurrence distance between identical events. The distances between, say, the up-going transitions for *all* repetitive signals of the circuit in Figure 1a are the same (equal to 10) after some initial period. This distance gives the *cycle time* of this circuit. In general, for more complex circuits it is not that simple, though.

From the timing diagram, we can further calculate the *average occurrence distance* up to a certain event, which is the occurrence time of the event divided by the number of times the event has occurred. The sequence for, say, the up-going transitions of a is: 2, $\frac{13}{2} = 6\frac{1}{2}$, $\frac{23}{3} = 7\frac{2}{3}$, $\frac{33}{4} = 8\frac{1}{4}$, $\frac{43}{5} = 8\frac{3}{5}$, $\frac{53}{6} = 8\frac{5}{6}$, ... The asymptote of this sequence is 10 and equal to the cycle time of this circuit.

One can conclude that the cycle time of the circuit is equal to the average occurrence distance for any repetitive event in an *infinite* timing simulation. In order to obtain the cycle time from a finite timing simulation, we face the problem that there are no simple criteria to establish the asymptote from a finite part of the sequence. In the general case, the sequence does not have to be monotonic, but may "oscillate" asymptotically towards the cycle time. Hence, we must ensure that the sequence yields the asymptote in a finite number of steps, and be able to establish an upper bound on the effort.

We note that the average occurrence distances over a finite simulation depends on the origin ($t = 0$) of the timing simulation and the initialization of the circuit, but the asymptotic behavior is independent of these. Therefore, by *initiating* the timing simulation in specific events we are able to throw away the effects of the initial history. If, for example, the timing simulation is initiated such that everything concurrent with and before $a{\uparrow}^0$ is assumed to have happened in the past, then we would obtain this sequence of average occurrence distances for up-going a-transitions: 10, 10, 10, ..., which immediately gives us the cycle time of this circuit, see Figure 1d.

If the timing simulation is initiated in an event which is part of a critical cycle, like event $a{\uparrow}$ for our example, then the sequence of collected average occurrences will yield the cycle time as the maximum average occurrence time within a finite simulation corresponding to the length of a critical cycle. If, on the other hand, the event is not part of a critical cycle, like event b, then all the collected numbers are smaller than the cycle time.

This opens two questions: How do we ensure to initiate the timing simulation in an event which is part of a critical cycle? And how do we bound the length of the timing simulation?

With simple arguments, we see that given a *cut set* of the graph, i.e., a set of events that contains at least one event from any cycle in the Signal Graph, then one of these events will be part of the critical cycle.Moreover, the number of periods any cycle can cover, is bound by the size of this cut set. One of the cut sets for the Signal Graph is given by the set of events with a bullet on some of their input arcs, called the *border events*. This set is not necessarily minimal, but is very simple to obtain.

In the Signal Graph in Figure 1b, there are two border events, $a{\uparrow}$ and $b{\uparrow}$. We perform a timing simulation covering two periods from each of the two border events, and collect the average occurrence distance after each full period. Starting with event $a{\uparrow}$ we obtain values: $\frac{10}{1} = 10, \frac{20}{2} = 10$, and with $b{\uparrow}$: $\frac{8}{1} = 8, \frac{18}{2} = 9$. The cycle time of the circuit equals the maximum of this set, $\Theta = 10$.

With this informal introduction in mind, we describe the derivation of the algorithm more formally.

III Signal Graphs

In this section, we describe the model for our analysis, Signal Graphs (following [9]), and its extension to Timed Signal Graphs.

A Basic definitions and properties

A *Signal Graph* is a tuple $\langle \mathcal{A}, I, \rightarrow, M, O\rangle$, where $\mathcal{A}$ is a set of events , $\mathcal{I} \subseteq \mathcal{A}$ is a set of initial events, and $\rightarrow \subseteq (A \times A)$ is the *precedence* relation between events. A pair of events which belongs to the relation, we call an *arc*. $\mathcal{M} : \rightarrow \longrightarrow \{0, 1, 2, \ldots\}$ is an initial marking function which assigns an initial number of tokens (the initial *activity*) to the arcs of a Signal Graph. $\mathcal{O}$ is a set of disengageable arcs, i.e. arcs which influence the execution a finite number of times only. We further distinguish the events in the Signal Graph, which are repetitive. They belong to the set $\mathcal{A}_r \subseteq \mathcal{A}$.

Example 1: The Signal Graph for the circuit in Figure 1a is shown in Figure 2a. $\mathcal{A}_r = \{a{\uparrow}, b{\uparrow}, c{\uparrow}, a{\downarrow}, b{\downarrow}, c{\downarrow}\}$, $\mathcal{I} = \{e{\downarrow}\}$ and $\mathcal{A} = \mathcal{A}_r \cup \mathcal{I} \cup \{f{\downarrow}\}$. ◁

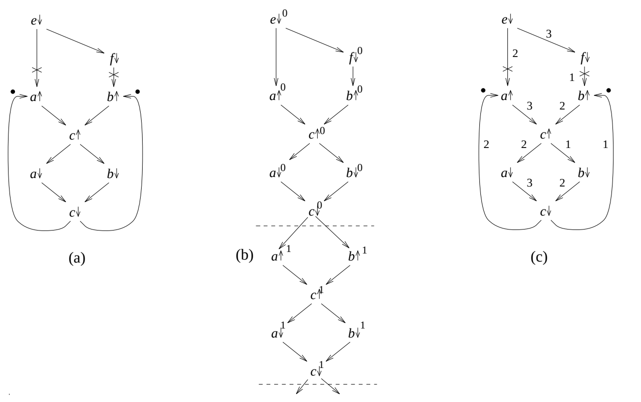

Figure 2: Signal Graph (a), unfolding (b) and Timed Signal Graph (c).

The execution model of a Signal Graph is similar to that of Marked Graphs. An event is *enabled* and can *occur*, when all its in-arcs (arcs from direct predecessors) have a positive activity. The result of such an occurrence is that the activity of all in-arcs are reduced by one, while the activity of all out-arcs are increased by one. For example, if the two arcs, $e \rightarrow g$ and $f \rightarrow g$, exist in the Signal Graph, then event g can occur after both events, e and f, has occurred only. In other words: Signal Graphs allow one to represent the behavior of concurrent systems with *AND-causality*, i.e., any event can occur after *all* of its predecessors have occurred. Neither *OR*-causality, nor non-deterministic choice is considered.

A Signal Graph is *bounded*, if in any execution the activity of the arcs in the Signal Graph are bounded by an upper limit. If this upper bound is equal to one, a Signal Graph is called *safe*. An event e is *reachable*, iff e appears in some feasible sequence of events. A Signal Graph is *live*, iff all events are reachable. In an acyclic Signal Graph, event e *precedes* event f, $e \Rightarrow f$, iff in every feasible sequence containing f, event e is present before f. Further, the events e and f of an acyclic Signal Graph are *concurrent*, $e \parallel f$, iff they are both reachable and $e \not\Rightarrow f$ and $f \not\Rightarrow e$. The notions of precedence and concurrency may be extended to cyclic Signal Graphs through their *unfolding*, which is discussed in the next section.

We restrict ourselves to Signal Graphs which obey the following properties:

- ∘ *Connected* (i.e. the underlying directed graph is strongly connected) and *bounded*.
- ∘ *Initially-safe*, i.e., the initial marking function, $\mathcal{M}$, is boolean: $\mathcal{M} :\rightarrow \longrightarrow \{0, 1\}$. Note, that this does not imply that the graph is safe. Any initially-non-safe graph can be transformed into an equivalent initially-safe one.
- ∘ Any unreachable events are considered to be redundant and the analysis is performed on the live subset of the Signal Graph only.
- ∘ There are no repetitive events before disengageable arcs. This is one of the properties of well-formedness [9].

Neither of the three latter restrictions does limit the descriptional power of Signal Graphs.

B Unfolding of Signal Graphs

In general, a Signal Graph contains both a non-repetitive and a cyclic part. A Signal Graph *unfolding* [9] is an acyclic process in which all events are non-repeated and correspond to single *instantiations* of events in the original Signal Graph.

An unfolding can be divided into *periods*. The first period contains the first instantiations of all events in the Signal Graph, the second period the second instantiations of the repeated events, etc. We will denote the i'th instantiation of an event, $e \in \mathcal{A}_r$, with $e^i,\ i \geq 0$.

All cyclic Signal Graph processes are quasi-periodic, which means that in the unfolding after a finite number of periods all succeeding periods will follow a fixed graph pattern.

Example 2: Figure 2b shows the two first periods of the unfolding of the Signal Graph from Figure 2a. $\triangleleft$

C Timed Signal Graphs

In order to perform timing analysis, we expand the definition of a Signal Graph to include timing information. Each of the arcs in a *Timed Signal Graph* is labelled with a delay, $\tau \in [0, +\infty)$. In the interpretation of Timed Signal Graphs we add delays to the execution. In the timing domain, AND-causality of Signal Graphs corresponds to the *MAX* execution model. For example, if there are two timed arcs, $e \overset{\tau_{eg}}{\rightarrow} g$ and $f \overset{\tau_{fg}}{\rightarrow} g$, in the Timed Signal Graph, and e and f have occurred at the moments of time, t_e and t_f, then g occurs at the moment $t_g = max\{t_e + \tau_{eg}, t_f + \tau_{fg}\}$. Further interpretation of Timed Signal Graphs will be given by the *timing simulation function* in the next section.

IV Timing simulation

In this section, we introduce the notions of timing simulation and event-initiated timing simulation, and then define the cycle time of a system.

A Timing simulation of a Signal Graph

We wish to be able to determine the *occurrence time* of an event in an execution of the Signal Graph. For this we use the *timing simulation* of an unfolded Timed Signal Graph, which is defined as

$$t(f) = \begin{cases} 0 & \text{if } f \in \mathcal{I}_u \\ \max\{t(e) + \tau \mid e \overset{\tau}{\rightarrow} f\} & \text{otherwise} \end{cases},$$

where $\mathcal{I}_u$ is the set of initial events of the unfolding. It consists of the events from $\mathcal{I}$ plus the repetitive events which have initially active in-arcs only.

Example 3: The initial part of the timing simulation of the Timed Signal Graph introduced in Figure 2c is shown in the following table:

$event$	$e{\downarrow}^0$	$f{\downarrow}^0$	$a{\uparrow}^0$	$b{\uparrow}^0$	$c{\uparrow}^0$	$a{\downarrow}^0$	$b{\downarrow}^0$	$c{\downarrow}^0$	$a{\uparrow}^1$	$b{\uparrow}^1$	$c{\uparrow}^1$	...
$t(event)$	0	3	2	4	6	8	7	11	13	12	16	...

The occurrence time of, say, $a{\downarrow}^0$ is calculated as:
$t(a{\downarrow}^0) = \max(\tau_{e{\downarrow}a{\uparrow}} + \tau_{a{\uparrow}c{\uparrow}}, \tau_{e{\downarrow}f{\downarrow}} + \tau_{f{\downarrow}b{\uparrow}} + \tau_{b{\uparrow}c{\uparrow}}) + \tau_{c{\uparrow}a{\downarrow}} = 8.$ $\triangleleft$

B Event-initiated timing simulation

In Section II, we demonstrated the use of initiating the timing simulation in specific events. Let us define the *event-initiated timing simulation* as

$$t_g(f) = \begin{cases} 0 & \text{if } f = g \text{ or } g \not\Rightarrow f \\ \max\{t(e) + \tau \mid (e = g \vee g \Rightarrow e) \wedge e \overset{\tau}{\rightarrow} f\} & \\ & \text{otherwise} \end{cases}$$

The intuition behind this definition is as follows: the g-initiated timing simulation is free of past-history caused by all events preceding and concurrent with g. These events are assigned the occurrence time zero, and all out-arcs for events concurrent to g are neglected. In general, this does *not* correspond to a simple linear shift of the timing simulation for all events with g as a predecessor, as the relative timing between the initiating event and events concurrent to it has potentially been changed.

Example 4: Assume that we wish to perform a $b{\uparrow}^0$-initiated timing simulation for the unfolding of the Timed Signal Graph from Figure 2c. The reachability set *without* $b{\uparrow}^0$ is: $\{e \mid b{\uparrow}^0 \not\Rightarrow e\} = \{f{\downarrow}^0, e{\downarrow}^0, a{\uparrow}^0\}$, therefore $t_{b{\uparrow}^0}(e{\downarrow}^0) = t_{b{\uparrow}^0}(f{\downarrow}^0) = t_{b{\uparrow}^0}(a{\uparrow}^0) = 0$. The reachability set *from* $b{\uparrow}^0$ is: $\{e \mid b{\uparrow}^0 \Rightarrow e\} = \{b{\uparrow}^0, c{\uparrow}^0, a{\downarrow}^0, b{\downarrow}^0, c{\downarrow}^0, a{\uparrow}^1, b{\uparrow}^1, c{\uparrow}^1, \ldots\}$. The initial part of the $b{\uparrow}^0$-initiated timing simulation is given by the following table:

$event$	$b{\uparrow}^0$	$c{\uparrow}^0$	$a{\downarrow}^0$	$b{\downarrow}^0$	$c{\downarrow}^0$	$a{\uparrow}^1$	$b{\uparrow}^1$	$c{\uparrow}^1$	$\ldots$
$t_{b{\uparrow}^0}(event)$	0	2	4	3	7	9	8	12	$\ldots$

◁

The following proposition demonstrates the close relation between the timing simulation and the longest path between two events in a Signal Graph.

Proposition 1: If event e_0 precedes event e_k, i.e. $e_0 \Rightarrow e_k$, then there exist at least one path of events, $(e_0, \ldots, e_k)$, connected pairwise with arcs, $e_i \xrightarrow{\tau} e_{i+1}$, such that

$$\textstyle\sum_{i=0}^{k-1} \tau_{e_i e_{i+1}} = t_{e_0}(e_k).$$

Further, there are no paths for which the sum is larger than $t_{e_0}(e_k)$. ◇

We will in the following use a special "cyclic" case of this proposition with $e_0 = e^i$ and $e_k = e^j$, which demonstrates a useful duality between an event-initiated timing simulation and the longest path between two instantiations of the same event, e^i and e^j, the so-called unfolded cycles described in Section V.

C Average occurrence distance

From the timing simulation we can define the *average occurrence distance* of an event, $e \in \mathcal{A}_r$, after i periods as follows:

$$\delta(e^i) = t(e^i)/(i+1).$$

For event-initiated timing simulations with the initial event being repetitive, we are especially interested in the average occurrence distance of subsequent instantiations of the initiating event:

$$\text{For } j > i \geq 0: \quad \delta_{e^i}(e^j) = t_{e^i}(e^j)/(j-i)$$

D Cycle time of a system

If we observe the occurrence times for consecutive instantiations of some repetitive event in a timing simulation, then after a finite initial period the event will occur following a fixed repeated pattern in time. This pattern is finite, but can cover several instantiations of the event, corresponding to several periods of the unfolding. The infinite repetition of this finite pattern cause the effect of the initial part to become arbitrary small when the timing simulation becomes long enough.

We define the *cycle time of an event* as the average occurrence distance of instantiations of the event for an infinite timing simulation: $\theta_e = \lim_{i\to\infty} \delta(e^i)$.

Proposition 2: All repetitive events in a connected and bounded Timed Signal Graph have the same cycle time:

$$\forall e, f \in \mathcal{A}_r: \quad \theta_e = \theta_f \qquad ◇$$

Therefore, we can define the *cycle time of a Timed Signal Graph* to be equal to the cycle time of any of the repetitive events:

$$\forall e \in \mathcal{A}_r: \quad \Theta = \lim_{i\to\infty} \delta(e^i).$$

E Using the infinite timing simulation to calculate the cycle time

The aim of this section is to prove that the cycle time can be calculated by the *maximum* operation.

Any arc between instantiations of repetitive events in a period of unfolding, P_i, will also be present between instantiations in all succeeding periods, P_{i+k}. This, we can utilize to show that a kind of a "triangular inequality" rule for the occurrence time of event instantiations is valid (see Figure 3). This implies the following proposition:

Proposition 3: For an e^i-initiated timing simulation, the occurrence time of a later instantiation of an event, $t_{e^i}(e^k)$, $k > i$, is larger than or equal to the sum of any combination of occurrence times for instantiations with occurrence indices between i and k and a total occurrence period equal to $k - i$:

$\forall e \in \mathcal{A}_r, \forall i \geq 0, \forall k > i:$

$$t_{e^i}(e^k) \geq \max\{t_{e^i}(e^j) + t_{e^i}(e^{i+k-j}) \mid i < j < k\} \qquad ◇$$

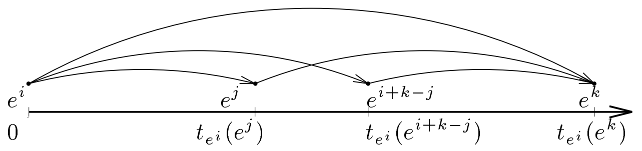

Figure 3: "Triangular inequality": $t_{e^i}(e^{i+k-j}) = t_{e^j}(e^k) \leq t_{e^i}(e^k) - t_{e^i}(e^j)$.

With the help of Proposition 3, we show that the cycle time of a Signal Graph is equal to the maximum average occurrence distance between successive instantiations of the initiating event in event-initiated timing simulations performed for *all* repetitive events:

Proposition 4: The cycle time of a circuit is given by:

$\forall e \in \mathcal{A}_r, \forall i \geq 0, \forall j > i:$

$$\Theta = \max\{t_{e^i}(e^j)/(j-i)\} = \max\{\delta_{e^i}(e^j)\} \qquad ◇$$

Proposition 4 give us a way to find the cycle time of a Signal Graph. For each of the repetitive events in the Timed Signal Graph we perform a timing simulation and calculate the average occurrence distance for each new occurrence of the initiating event. The cycle time will be given by the maximum of all these occurrence distances. The problem is, that the search space is infinitely large. For all events in the unfolding, we need to simulate infinitely long to ensure, we have found the maximum average occurrence distance. We need some criteria to limit the search space.

V The cycle time and graph cycles

In this section, we discuss the correspondence between cycle time and the cycles in the graph. This will give us the knowledge necessary to obtain the cycle time from timing simulations in a finite number of steps.

A Cycles in Signal Graphs

In a Timed Signal Graph with repetitive events there exist cycles. A cycle is a closed path formed by the set of events, $C = \{e_0, e_1, \ldots e_{n-1}\} \subseteq \mathcal{A}_r$, $n > 0$, connected by the set of arcs: $\{e_i \to e_{(i+1)\,\mathrm{mod}\,n} \mid 0 \leq i < n\}$.

The *length* of the cycle is given by:

$$\overline{C} = \sum_{i=0}^{n-1} \tau_{e_i e_{(i+1) \bmod n}}$$

The cycle is *simple* if no event in it occur more than once.

In the unfolded Signal Graph, a cycle is represented by an equivalent *unfolded cycle*, which is a path between two instantiations e_0 and e_n of the same event formed by the set of events, $C_u = \{e_0, e_1, \ldots, e_n\} \subset \mathcal{A}_u$, and the set of arcs connecting the events: $\{e_i \to e_{i+1} \mid 0 \leq i < n\}$. The unfolded cycle is simple if it contains one instantiation of the same event only.

We will in the following represent a cycle (or an unfolded cycle) by the set of participating events, C (or C_u), and let the connecting arcs be implied. We denote the set of all simple cycles $\mathcal{C}$ and the (infinite) set of all unfolded simple cycles $\mathcal{C}_u$.

As a direct consequence of Proposition 1 we can formulate the following "cyclic" case of that proposition:

The length of the longest unfolded cycle between two instantiations of the same event, e^i and e^j, $i < j$, is equal to the event-initiated timing simulation $t_{e^i}(e^j)$.

To each of the cycles we associate a parameter, called the *occurrence period* of the cycle, denoted by ε, equal to the number of Signal Graph periods the unfolded cycle covers. Let us assume that $C_u = \{a^i, \ldots, a^j\}$, $j > i$, then $\varepsilon = j - i$.

From the length of a cycle and its occurrence period we define the *effective length* of the cycle as $\overline{C}/\varepsilon$, i.e. the length of the cycle per period it covers.

Example 5: The Timed Signal Graph in Figure 2c contains four simple cycles, $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$:
$C_1 = \{a{\uparrow}, c{\uparrow}, a{\downarrow}, c{\downarrow}\}$, $C_2 = \{a{\uparrow}, c{\uparrow}, b{\downarrow}, c{\downarrow}\}$, $C_3 = \{b{\uparrow}, c{\uparrow}, a{\downarrow}, c{\downarrow}\}$, and $C_4 = \{b{\uparrow}, c{\uparrow}, b{\downarrow}, c{\downarrow}\}$. The length of the first cycle is: $\overline{C_1} = 10$, while the length of the fourth cycle $\overline{C_4} = 6$. The unfolded cycles equivalent to C_1 are given by: $C_{1,i} = \{a{\uparrow}^i, c{\uparrow}^i, a{\downarrow}^i, c{\downarrow}^i, a{\uparrow}^{i+1}\}$ for $i \geq 0$. All the cycles have an occurrence period equal to one. ◁

B The cycle time and simple cycles

As there is a duality between event-initiated timing simulations and unfolded cycles (Proposition 1), the cycle time can be found both as the limit of the average occurrence distance of any event (by definition) and as the longest effective length of *all* unfolded cycles (follows from Proposition 4). In the next proposition, we show that it is necessary to consider *simple* cycles only.

Proposition 5: The effective length of any unfolded non-simple cycle is no greater than the largest of the effective lengths of the simple cycles from which it has been combined:
$\overline{C}/\varepsilon \leq \max\{\overline{C_i}/\varepsilon_i \mid C_i \subseteq C\}$ ◇

It follows directly from this and Propositions 1 and 4 that the cycle time of the circuit, can be found by checking all simple cycles in the Signal Graph:

$$\Theta = \max\left\{ \frac{\overline{C_i}}{\varepsilon_i} \middle| C_i \subseteq \mathcal{C} \right\} \quad ,$$

The cycles, for which the maximum is actually reached are called *critical cycles.*

Example 6: The cycle time of the Timed Signal Graph in Figure 2c is given by:

$$\begin{aligned} \Theta_{ex1} &= \max\left\{ \frac{\overline{C_1}}{\varepsilon_1}, \frac{\overline{C_2}}{\varepsilon_2}, \frac{\overline{C_3}}{\varepsilon_3}, \frac{\overline{C_4}}{\varepsilon_4} \right\} \quad (\varepsilon_i = 1) \\ &= \max\{10, 8, 8, 6\} \;=\; 10 \end{aligned}$$

◁

However, the number of cycles in a graph may be exponential in the number of arcs in the graph. Consequently, an algorithm searching for and checking all cycles has potentially exponential execution time. Instead, we use our knowledge about critical cycles to find a halt criteria for the timing simulations discussed in Section IV. We discuss this in the following section.

VI Using the finite timing simulation to obtain the cycle time

We show in this section that for finding the cycle time,

- a few of the repetitive events need to be considered only, and these events can be easily selected from the Signal Graph;
- a simple criterion to bound the length of the timing simulation does exist.

A Cut sets

A set of events is a *cut set* if at least one event from any cycle in the Signal Graph are represented in the set.

For our purposes a very convenient cut set is the *border set* which is the set of events with an initialization mark on any of the input arcs. This set is by definition a cut set for a live Signal Graph: *For a Signal Graph to be live, all cycles contain an initial mark* [5]. The set is called the "border set" because in the unfolding the instantiations of the events in the set establishes a border between the periods of the unfolding. All arcs with an initial marking cross the border between two periods.

A cut set is called a *minimum cut set* if it contains the fewest possible elements.

Example 7: In the Signal Graph from Example 1, $\{a{\uparrow}, b{\uparrow}\}$ is the border set. Other cut sets exist, e.g. $\{c{\uparrow}\}$ or $\{a{\downarrow}, b{\downarrow}\}$. $\{c{\uparrow}\}$ and $\{c{\downarrow}\}$ are minimum cut sets; the rest are not minimum. ◁

B Limiting the timing simulations

Instead of checking all cycles in the Signal Graph, we use event-initiated timing simulations to find the cycle time and the corresponding longest cycle(s).

We establish an upper bound on the number of periods a simple unfolded cycle can cover in the Timed Signal Graph.

Proposition 6: The maximum number of unfolded Signal Graph periods a simple cycle can cover, *the maximum occurrence period of any cycle*, $\varepsilon_{\max}$, is bound by the size of a minimum cut set. ◇

All cycles in the Signal Graph do by definition include at least one member from any cut set. We can therefore limit the timing simulations to be initiated from members of a cut set instead of all events in the Signal Graph. In our implementation of the algorithm, we have chosen not to search for a minimum cut set, as this is a complex optimization task. Instead we use as a cut set the border set which is given directly with the Signal Graph.

We present two key propositions, which form the base for the algorithm for performance analysis. They are used to find the cycle time of the Timed Signal Graph, and sort out which members of a given cut set belongs to a critical cycle.

Proposition 7: Let event e belong to a critical cycle, and k be the number of events in a minimum cut set. Then the cycle time of a Timed Signal Graph can be found as:

$$\Theta = \max\{\delta_{e^0}(e^i) \mid 0 < i \leq k\}. \qquad \diamond$$

Proposition 8: If an event, e, is *not* part of a critical cycle, all average occurrence distances calculated for any of the succeeding instantiations of e will be smaller than the cycle time.

$$\forall n: \ \max\{\delta_{e^0}(e^i) \mid 0 < i \leq n\} < \Theta$$

$\diamond$

In spite of Proposition 8, it is still true that $\lim_{i\to\infty} \delta_{e^0}(e^i) = \Theta$ for events, e, which are not part of a critical cycle. That is, these events demonstrate asymptotic behavior with the cycle time as an asymptote. They will, however, never reach this asymptotic value. Figure 4 shows a typical example of the asymptotic behavior of $\delta_{e^0}(e^i)$ for events *on* a critical cycle and events *off* a critical cycle.

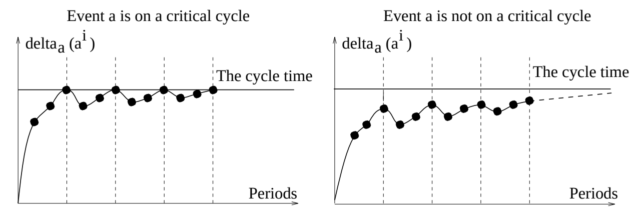

Figure 4: Asymptotic behavior of $\delta_{e^0}(e^i)$ for events *on* a critical cycle (a) and events *off* a critical cycle (b).

When the border events belonging to a critical cycle have been found, we can use Proposition 1 to "backtrack" the timing simulation that yielded the critical cycle in order to obtain all events in the critical cycle.

VII THE ALGORITHM

In order to find the cycle time, we need to perform a timing simulation from any event which is part of a critical cycle (Proposition 7). All cycles in the Signal Graph do by definition include at least one member from the set of "border events" (as a particular case of a cut set). Therefore, we can perform an event-initiated timing simulation for each of the b events from the border set, and be sure that among these, there is an event belonging to the critical cycle. Also, due to Proposition 7, it is sufficient to perform each timing simulation for b instantiations of the initiating border event. The cycle time is found as the maximum of the b^2 average occurrence distances which were collected during the timing simulations. We use Propositions 7 and 8 to select which border events are part of a critical cycle.

The skeleton of the algorithm

1. Create the Timed Signal Graph.

2. Identify the "border events".

3. For each of the b border events do:

   ◦ Perform a timing simulation corresponding to b periods of the Timed Signal Graph starting in the border event.

   ◦ For each new occurrence of the initiating event of the current timing simulation, calculate the corresponding average occurrence distance.

4. The largest average occurrence distance corresponds to the cycle time of the circuit.

5. "Backtrack" the corresponding critical cycle(s) from border events yielding the cycle time.

Let us estimate the complexity of the algorithm. Assume that m is the number of arcs in the Timed Signal Graph. The complexity of performing one timing simulation is dictated by the number of arcs in the graph unfolded into b periods, which is less than or equal to $b \times m$. Since we need to perform a timing simulation for each of the border events, the overall complexity is estimated to be $O(b^2 \times m)$. Note, that the number of border events b is typically $\ll m$, such that the algorithm will often demonstrate linear complexity from the size of the Timed Signal Graph specification.

VIII APPLICATION TO CIRCUIT ANALYSIS

In this section, we are looking at a specific application of the performance analysis algorithm: the analysis of asynchronous circuits. Given a net-list for a circuit and an initial state, we are able to determine the expected performance for the circuit.

A Signal Graphs and circuits

Our focus has been on a special class of asynchronous circuits, the *distributive circuits*, which operates correctly independent of any variation of delays in circuit elements. The distributivity property is sufficient (but not necessary) to guarantee the speed-independence of a circuit behavior.

It has been shown that any correct Signal Graph can be implemented as a distributive circuit, and any distributive circuit can be represented with a Signal Graph [9].

In order for a Signal Graph to be implementable as a circuit, we need a couple of additional conditions specific for the behavior of signal transitions:

Switch-over correctness: In any sequence of signal changes the up-going and down-going transitions of a signal must alternate.

No auto-concurrency: There must be no concurrent changes of the same signal.

A Signal Graph may contain several events corresponding to the same signal (*multiple* events), e.g., there may be several occurrences of $a\!\uparrow$. We will consider each of these occurrences as separate events with different names, e.g. $a1\!\uparrow$, $a2\!\uparrow$, etc. Note, that *delays* for the same signal can vary from one event to another, and delays associated with different in-arcs of the same event can differ as well. This allow us to deal with individual input-output characteristics of a transistor-level gate implementation.

B Derivation and analysis of a Signal Graph

In [9] an efficient algorithm is presented that, given a circuit and an initial state, verifies that the circuit is distributive, and extracts a Signal Graph specifying the circuit behavior.

If the circuit is distributive, the algorithm produces the complete Signal Graph; otherwise it finds the states, where a violation of distributivity occurs, and produces the initial part of the Signal Graph which precedes this violation. This algorithm is implemented in the system TRASPEC, which is part of the FORCAGE 3.0 CAD tool for asynchronous circuits.

Our cycle time algorithm is running under UNIX. The analysis of, for example, a Signal Graph with 66 events and 112 arcs, which describes the gate level behavior of an asynchronous stack with constant response time, takes 74 CPU milliseconds on a DEC 5000.

C C-element oscillator

We obtain the cycle time for the circuit specified by its Timed Signal Graph in Figure 2c through timing simulation.

The border set is $\{a\uparrow, b\uparrow\}$, i.e., we perform the event initiated timing simulations over two periods. As a minimum cut set consists of one element (e.g. $\{c\uparrow\}$), one period is needed only.

$event$	$a\uparrow^0$	$b\uparrow^0$	$c\uparrow^0$	$a\downarrow^0$	$b\downarrow^0$	$c\downarrow^0$	$a\uparrow^1$	$b\uparrow^1$	…	$c\downarrow^1$	$a\uparrow^2$	$b\uparrow^2$
$t_{a\uparrow^0}(event)$	0	0	3	5	4	8	10	9	…	18	20	19
$\delta_{a\uparrow^0}(a\uparrow^i)$	–	–	–	–	–	–	10	–	…	–	10	–
$t_{b\uparrow^0}(event)$	0	0	2	4	3	7	9	8	…	17	19	18
$\delta_{b\uparrow^0}(b\uparrow^i)$	–	–	–	–	–	–	–	8	…	–	–	9

The cycle time is given by:
$\Theta = \max\{\delta_{a\uparrow^0}(a\uparrow^1), \delta_{a\uparrow^0}(a\uparrow^2), \delta_{b\uparrow^0}(b\uparrow^1), \delta_{b\uparrow^0}(b\uparrow^2)\} = 10.$
The critical cycle is: $a\uparrow \rightarrow c\uparrow \rightarrow b\downarrow \rightarrow c\downarrow \rightarrow a\uparrow$

To illustrate the behavior of timing simulations initiated in an event which is not on a critical cycle, we observe the average occurrence distances obtained by the infinite $b\uparrow^0$-initiated timing simulation: $\max\{\delta_{b\uparrow^0}(b\uparrow^i) \mid i > 0\} = \max\{8, 9, 9\frac{1}{3}, 9\frac{1}{2}, 9\frac{3}{5}, \ldots\} \approx \lim_{i\rightarrow\infty} \delta_{b\uparrow^0}(b\uparrow^i) = 10$

D Muller ring

In the following example, we analyze a circuit where the critical cycle covers more than one period of the unfolded Timed Signal Graph. The circuit is a Muller pipeline with five C-elements, where the two ends of the pipeline are connected to form a ring. The ring is initialized such that it contains one "data token", represented by a signal value equal to 1. A diagram of the circuit together with the corresponding Signal Graph is shown in Figure 5. Initially the output of the last C-element is high, while the others are low. We assume the delay of both the C-elements and the inverters to be equal to 1, i.e., all arcs in the Timed Signal Graph are assigned the delay 1.

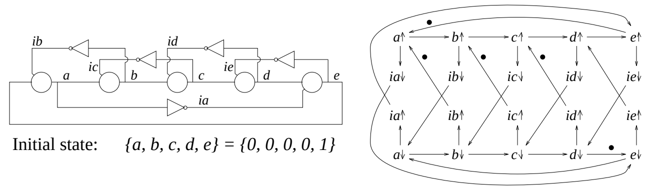

Figure 5: Diagram and Signal Graph for Muller ring with five elements.

The Signal Graph contains four border events: $a\uparrow$, $b\uparrow$, $c\uparrow$ and $e\downarrow$. For each of the border events, we perform an event-initiated timing simulation covering four periods of the unfolding. After each period, we calculate and collect the average occurrence distance. As the circuit is symmetric for the four border events, the four timing simulations yield the same result. In the following table, we show the results from the $a\uparrow$-initiated timing simulation. To illustrate the periodic behavior of the occurrence distances, we have extended the timing simulation to cover ten periods instead of the required four:

i	1	2	3	4	5	6	7	8	9	10
$t_{a\uparrow^0}(a\uparrow^i)$	6	13	20	26	33	40	46	53	60	66
$\Delta_{a\uparrow^0}(a\uparrow^i)$	6	7	7	6	7	7	6	7	7	6
$\delta_{a\uparrow^0}(a\uparrow^i)$	6	6.5	6.67	6.5	6.6	6.67	6.57	6.63	6.67	6.6

The cycle time of the circuit is found to be:
$\Theta_{ring5} = \max\{\delta_{a\uparrow^0}(a\uparrow^i) \mid 0 < i \leq 4\} = \frac{20}{3}\ (\approx 6.67).$

IX Conclusion

We have presented an efficient algorithm for determining the cycle time of a circuit from its Timed Signal Graph representation. The main characteristic of the algorithm is that it is based on the conceptually very simple frameworks of the unfolding of the Timed Signal Graph and timing simulations.

Our work has been focussed on circuit analysis. We view the Timed Signal Graph as being a circuit specification or, alternatively, as being a behavioral representation of an existing circuit. To this end we apply as a preliminary step the algorithm described in [9] to extract the Signal Graph from a circuit description. The applications may, however, be generalized for the analysis of other concurrent systems which can be represented by an event model similar to the Signal Graph representation.

Acknowledgements

We are grateful to Steve Burns, Jørgen Staunstrup, Jens Kargaard Madsen, Alex Kondratyev and Luciano Lavagno for discussions of this paper.

References

[1] F. Baccelli, G.Cohen, G.J.Olsder, and J.P. Quadrat. *Synchronization and Linearity*. John Wiley and Sons, 1992.

[2] S. M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, California Institute of Technology, December 1990. (TR Caltech-CS-TR-91-01).

[3] C.D.Nielsen and M.Kishinevsky. Performance analysis based on timing simulation. ID-TR: 1993-125, Technical University of Denmark, Nov. 1993 (ftp ftp.id.dth.dk, pub/TR/1993/ID-TR:1993-125.ps.Z).

[4] T.-A. Chu. *Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications.* PhD thesis, MIT, June 1987.

[5] F. Commoner, A.W. Holt, S.Even, and A.Pnueli. Marked directed graphs. *J. of Comp. and Syst. Sci.*, 5:511–523, 1971.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1990.

[7] J. Gunawardena. Timing analysis of digital circuits and the theory of min-max functions. In *Proc. of the ACM Int. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, 1993.

[8] M. Hartmann and J. Orlin. Finding minimum cost to time ratio cycles with small integral transit times. *Networks*, 23:567–574, August 1993.

[9] M. A. Kishinevsky, A. Y. Kondratyev, A. R. Taubin, and V. I. Varshavsky. *Concurrent Hardware. The Theory and Practice of Self-Timed Design.* John Wiley and Sons Ltd., 1994.

[10] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for synthesis and testing of asynchronous circuits.* Kluwer Academic Publishers, 1993.

[11] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[12] P. Vanbekbergen, F. Catthoor, G. Goossens, and H. De Man. Optimized synthesis of asynchronous control circuits from graph-theoretic specifications. *IEEE Tr. on Computer-Aided Design*, pp. 1426–1438, Nov. 1992.

[13] N. Young, R. Tarjan, and J. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21:205–221, 1991.