Verification of Speed-Dependences in Single-Rail Handshake Circuits

Radu Negulescu
University of Waterloo
Waterloo, Ontario, Canada
radu@maveric.uwaterloo.ca

Ad Peeters
Philips Research Laboratories
Eindhoven, The Netherlands
apeeters@natlab.research.philips.com

Abstract

A way to reduce the cost (area) or increase the performance of asynchronous circuits is to make timing assumptions that go beyond the isochronic fork. This, however, results in circuits that are not speed-independent. Such timing assumptions often boil down to imposing that, of two circuit paths that start at the same point, one path is faster than the other. We call speed-dependences of this form chain constraints, and we handle them as processes in a metric-free formalism.

This paper applies chain constraints to verify single-rail handshake circuits in the context of their timing assumptions, and to evaluate safety margins for delay fluctuations. We discuss the lessons learned, including decomposition and weakening of extended isochronic fork assumptions, usage of CMOS cell models in the presence of hazards, and correlations between our discrete-state results and analog simulations.

1 Introduction

Single-rail handshake circuits [Pe96] are a type of asynchronous circuits with proven industrial viability. For efficiency, single-rail handshake circuits often use extended isochronic forks as defined in [vBHP95]. This design style allows for substantial gains in speed, area, and power economy, but makes local compromises to speed-independence, by causing the circuits to rely on relative delays in cells and wires.

To verify a circuit that relies on delay constraints, one might (i) size the component delays to meet the delay constraints and then verify the circuit with known bounds on component delays, or (ii) verify the circuit solely on the basis of given delay constraints, then size the component delay bounds to meet the delay constraints. Precise bounds on individual component delays can be calculated only after knowing transistor sizes and wire lengths; for this reason, let us call these approaches *post-layout* and *pre-layout* verification, respectively. Pre-layout verification can speed up the design cycle by avoiding optimizing delays in an incorrect circuit, by allowing migration to another implementation technology for a verified circuit, and by keeping the verification simple because a minimum of information is used.

A technique for pre-layout verification was proposed in [Ne97]. There, unequal path delay assumptions are formalized as 'chain constraints,' represented as processes in a metric-free process space from [Ne95]. Such processes can be coupled and compared to other processes representing circuit specifications or components. Verification and diagnosis are performed by a tool based on binary decision diagrams (BDDs). The formalism and tool are briefly described in Section 2.

Here we apply the technique from [Ne97] to several circuits from [Pe96]. We also report our experience with the following problems, which are encountered in these case studies but were not discussed in [Ne97]:

- significance of hazards (are they true faults? can they be ignored?);
- models for the behavior of certain CMOS cells in the presence of hazards;
- comparisons to Spice-like, analog simulations of dynamical system circuit models;
- extraction of chain constraints from extended isochronic fork assumptions;
- interpretations of delay constraints on analog waveforms;
- the importance of threshold spreads for the implementation of chain constraints;
- assessment of timing reliability in the presence of chain constraints;
- dealing with data signals.

2 The Formalism Used

Process spaces are a simple theory that unifies the treatment of several types of systems and correctness criteria by abstracting the notion of 'execution.' Executions can be sequences of events, functions of time, etc., but, in the main theory, they are just elements of an arbitrary set $\mathcal{E}$. Process spaces were introduced in [Ne95], and additional explanations, examples, and a description of a related tool are available in [Ne97].

A process over set $\mathcal{E}$ is a pair (X, Y) of subsets of $\mathcal{E}$ such that $X \cup Y = \mathcal{E}$. A process represents a contract between a device and its environment, from the device point of view. A process (X, Y) splits $\mathcal{E}$ into subsets $X \cap Y$, $\overline{X}$, and $\overline{Y}$, whose executions are called *goals*, *errors*, and *rejects*, respectively. Executions in $X \cap Y$ are legal for both the device and the environment. Executions in $\overline{X}$ represent errors of the device. Finally, executions in $\overline{Y}$ represent bad behavior on the part of the environment, and, as such, can be rejected by the device.

In this paper, we use only the following operations on processes:

- Robustness is a unary predicate; a process is robust if it imposes no requirements on its environment.
- Refinement is a binary relation, written $p \sqsubseteq q$, meaning that 'process q is a satisfactory substitute for process p.'
- Reflection, denoted by $-$, is a unary operation. If p represents a contract between a device and its environment from the device point of view, $-p$ represents the environment viewpoint in the same contract.
- Product, written $\times$, is a binary operation, yielding a process that models a system of two processes operating 'jointly.'

Refinement is reflexive, transitive, and antisymmetric. Reflection is its own inverse. Product is commutative, associative, and idempotent. Furthermore, for processes p, q, and r,

$$p \sqsubseteq q \;\Rightarrow\; p \times r \sqsubseteq q \times r \,.$$

As described in [Ne95, Ne97], these properties enable us to break a refinement verification problem into several layers of refinement of partial specifications, and each layer into several modules, and to verify only one module at a time instead of verifying the overall problem in one piece.

By the definition in [Ne95], a process is robust if it has no reject executions. Moreover, for processes p and q,

$$p \sqsubseteq q \;\Leftrightarrow\; -p \times q \text{ is robust} \,.$$

Thus, one can detect violations of refinement as counter-examples to the robustness of $-p \times q$, i.e., as executions that are rejects for $-p \times q$. Such counter-example executions can be regarded as 'witnesses to failure' in the verified system.

In this paper, we use finite sequences of events as executions, and we interpret them as complete operations of a system rather than initial fragments of the operation. This interpretation captures not only illegal events, but also illegal stopping. This process space and its properties are closely related to the prefix-closed trace structure theory in [Di89], and also include a deadlock-freedom concern as in [Ve94]. Process spaces also have strong similarities to other 'trace-oriented' treatments of concurrency, notably to those in [Eb91], [Ho85], and [Jo92].

Manipulations of finite-word processes are implemented by a tool called FIREMAPS [Ne97]. The label stands for finitary and regular manipulation of processes and systems, and also refers vaguely to mapping firings of a Petri net. The tool uses a BDD library [BRB90] which offers basic routines for manipulating large Boolean functions. The operations and conditions implemented in FIREMAPS include (but are not limited to) the process space operations and conditions mentioned above, and constructors of models for circuit components and constraints, described below.

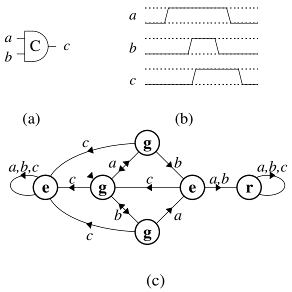


Figure 1: Example process: (a) C-element; (b) example waveform; (c) process representation (symbol '▶' indicates the initial state).

To illustrate the FIREMAPS models of circuit components, consider the C-element in Fig. 1 (a). A C-element copies at the output the value of the inputs if the inputs have the same value, and keeps the output constant otherwise. Waveforms are represented by finite sequences of actions corresponding to signal transitions, such as $abcbac$ for the waveform in Fig. 1 (b). We sometimes indicate that a certain action represents a rising or falling transition, as in $a\!\uparrow b\!\uparrow c\!\uparrow b\!\downarrow a\!\downarrow c\!\downarrow$.

If all signals start low, the C-element is represented by the process in Fig. 1 (c), where **r**, **g**, and **e** stand for reject, goal, and error (executions that stop at a state marked **r** are rejects, executions that stop at **g** are goals, etc.). Illegal input events lead to a reject state that cannot be left, illegal output events lead to an error state that cannot be left. In contrast, the state where ab leads *can* be left, but is also marked as an error, making it illegal for the device to complete its operation by stopping there. In this example, we make it illegal for the environment to disable ongoing output transitions of the cell, thus executions such as abb are marked as rejects; other models for the C-element will be discussed later.

We build process models for CMOS cells directly from a Boolean description of the unstable states. For instance, it suffices to specify that the C-element has output c and that it is unstable if and only if its signals satisfy $\neg(a \oplus b) \wedge (a \oplus c)$, where $\wedge$ and $\oplus$ are conjunction and exclusive-or.

3 Modeling Issues

The modeling of digital circuits by means of finite-state processes is a well-studied issue in concurrency theory and asynchronous systems. We add to previous discussions by detailing the way we handle hazards, extended isochronic forks, and other circuit features, and by relating certain points of our discrete analysis to analog simulations.

3.1 Hazards

There are some variations of the CMOS cell models, because, in the presence of hazards, the behavior of a CMOS cell is not fully standardized. A hazard is a situation where an output transition is enabled and then disabled without being fired. For example, execution abb is a hazard for the C-element in Fig. 1. Depending on the implementation, and the relative timing of the first and second event on b, the C-element may stop after abb, may respond by a c transition, etc.

To handle various responses to hazards, we recast in process spaces several common models of gates, used for instance in [BS95], [Di89], and [La92]. Models that simply require the environment to avoid hazards, as in Fig. 1 (c), are called *hazard-intolerant* in [Ne97]. Models that ignore hazards, for instance by making $abba$ lead the C-element back to its initial state as in Fig. 2 (a), are called *inertial*[1]. Other models, which can 'store' up to k transitions after hazards, are called *k-bounded pure delay* models; we are interested in even values for k, especially $k = 2$. An example is shown in Fig. 2 (b), obtained by attaching an inertial buffer at the output of an inertial cell.

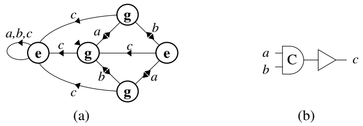


Figure 2: Alternative C-element models: (a) inertial model; (b) example of 2-bounded model.

Hazards do not necessarily constitute malfunctions of a circuit, but it is best to know whether they may occur, and to use more detailed models for the particular behaviors of those gates exposed to hazards. For this, we use hazard-intolerant models for verification, and, if hazards are found, we use inertial or 2-bounded pure delay models for diagnosis. This way, we apply pessimistic verification criteria, including hazard-freedom, but we insist on finding faults that are more tangible than the hazards.

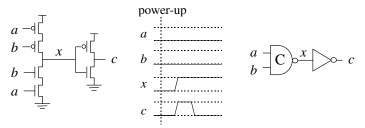
FIREMAPS has constructors for building both hazard-intolerant and inertial models from the unstable state representation in Section 2. The k-bounded models for even k can be obtained as products of other models, by inserting inertial buffers at the outputs of the CMOS cells, or by inserting inversions to obtain a slightly different behavior at initialization.

3.2 Initialization

If the power voltage is set up briskly, a C-element implementation like that in Fig. 3 (a) may issue an output pulse even if the inputs stay low, as in Fig. 3 (b). Such effects can be captured by inserting inversions instead of buffers at the output of a CMOS cell, as in Fig. 3 (c).

However, in practice the initial behavior of a circuit is treated separately from other aspects of its operation; special precautions are taken to ensure that the circuit reaches a desired stable state within a short time after power-up, and that the circuit is used only from that state on. If so, we assume in the analysis that the circuit is started in the desired stable state.

[1] Strictly speaking, the usual inertial models also include numerical bounds on gate delays; see e.g. [BS95].



(a) (b) (c)

Figure 3: Discussion of initial behavior: (a) possible transistor implementation of the C-element; (b) possible behavior after a brisk power-up; (c) alternative 2-bounded model.

Our CMOS cell models assume by default a (stable or unstable) initial state where all signals are low. To change the initial state, we use a language derivative operator similar to the 'after' operators in [Ho85], [Ve94], etc., forcing an odd number of actions for each signal that is assumed to start high. To re-initialize the model in Fig. 3 (c) to a, b, and c high, we can change the initial states of the cell processes as if sequence abc had occurred. Care must be taken not to use illegal events in the initialization sequences.

Undetermined initial states can also be modeled, using the process spaces 'meet' operator, which is implemented in FIREMAPS; but, for the reasons above, this is beyond the scope of our paper.

3.3 Specifications

The intended behavior of a circuit is specified in [Pe96] either by production rules like those in [Ma90] or by a form of CSP-like commands similar to those in [Eb91] and [vB93].

Production rules have the general form

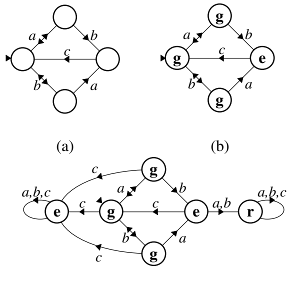
$$\begin{cases} U(a_1, a_2, \ldots) \to b\uparrow \\ D(a_1, a_2, \ldots) \to b\downarrow \end{cases}$$

meaning that output b of a cell is pulled up whenever function U of inputs is true, and is pulled down whenever D is true. Such specifications can easily be translated into a Boolean function that characterizes the unstable states:

$$U(a_1, a_2, \ldots) \wedge \neg b \vee D(a_1, a_2, \ldots) \wedge b$$

which is translated automatically into a finite-state process, as mentioned in Section 2. We normally use hazard-intolerant models for handshake specifications.

As in [Eb91], a command can specify a regular, prefix-closed, and non-empty language, represented by a state graph like that in Fig. 4 (a). This particular state graph represents the C-element from Fig. 1. We modify a state graph by declaring the 'stable' states to be goals and the



(a) (b)

(c)

Figure 4: Constructing a process from a command: (a) original state graph; (b) specifying unstable states; (c) process representation.

'unstable' states to be errors, similarly to the distinction between 'quiescent' and 'transient' nodes in [vB93]. The state machine of a process must be a complete automaton, to qualify all finite words over a given alphabet as either rejects, goals, or errors. For that, we add to the state graph the illegal input and output events: illegal outputs lead to an error state that cannot be left, thus making it the responsibility of the device to avoid any execution containing an illegal output; similarly, illegal inputs lead to a reject state that cannot be left. These transformations for the state graph in Fig. 4 (a) yield the process in Fig. 4 (c), the hazard-intolerant model of a C-element.

Specifications may contain references to 'guards,' or values of data signals. If that is the case, the state graph of the specification consists of several parts, one part for each possible value of the guard. These parts are connected, but fairly loosely. An example is the specification for the DO-component case study in Subsection 5.3.

3.4 Chain Constraints

In [Ne97], we also use processes to model relative delay assumptions of the form:

$$D_{b_1b_2} + \cdots + D_{b_{n-1}b_n} > D_{a_1a_2} + \cdots + D_{a_{m-1}a_m} \quad (1)$$

where $a_1, \ldots, a_m, b_1, \ldots, b_n$ are events such that a_1 is the same as b_1, and the D's are delays between events. Such a constraint, called a *chain constraint*, imposes that the b chain of events will not be completed before the a chain, unless one of the a or b actions involved occurs out of order and disrupts the chains. Instead of the expression in equation (1), we often use the shorthand

$$D_{b_1\ldots b_n} > D_{a_1\ldots a_m} \,. \quad (2)$$

Because of the 'disruption' case and another case where the chains stop midway, we could not deal with a chain constraint as just a linear inequality. (Some details to that effect are provided in [Ne97].) On the other hand, we can deal with chain constraints as processes. FIREMAPS provides special operators for building such a process from the two chains of events involved and from an initialization sequence specifying the circuit states where the constraint applies.

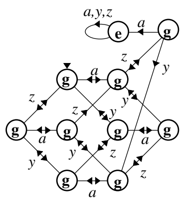


Figure 5: Process for $D_{z\uparrow a\downarrow} > D_{z\uparrow y\downarrow}$, assuming a and z start low and y starts high.

For example, the (rather simplistic) chain constraint $D_{z\uparrow a\downarrow} > D_{z\uparrow y\downarrow}$ would be represented by the process in Fig. 5. Here the direction of signal transitions is meaningful, indicating that the constraint applies from a state where z is low and a and y are high. This state, called the *base state* of the constraint, needs not be the initial state of the system. A cube-like set of states keeps track of the levels of the three signals involved. Assuming that initially a and z are low and y is high, executions such as $a\uparrow\, z\uparrow\, a\downarrow\, z\downarrow$ violate the constraint and are declared to be errors, thereby making it the responsibility of an imaginary constraint device to avoid such executions. Executions such as $a\uparrow\, z\uparrow\, y\downarrow$, which obey the constraint, or $a\uparrow\, z\uparrow\, z\downarrow$, which disrupt the chains of events, are not restricted by the constraint process, which qualifies them as goals.

3.5 Extended Isochronic Fork Assumptions

An extended isochronic fork assumption of depth n [vBHP95] imposes that the differences of the delays in circuit paths that start at a certain node and go through n layers of (inverting CMOS) gates must be smaller than delays in the next layer of gates. In a first approximation, let us ignore wire delays and rise and fall times. Wire delays can be handled by simply introducing buffer components between the gates of the circuit, while the rise and fall times of signal transitions are considered in Subsection 3.6. An extended isochronic fork assumption can be written as:

$$|(D_1 + \cdots + D_n) - (D'_1 + \cdots + D'_n)| < D'' \quad (3)$$

where $|\cdot|$ denotes the absolute value, $D_1, \ldots, D_n$ and $D'_1, \ldots, D'_n$ are the gate delays in the two paths, and D'' is the delay of a gate in the next layer.

Inequality (3) splits into:

$$D_1 + \cdots + D_n < D'_1 + \cdots + D'_n + D'' \quad (4)$$
$$\text{and} \quad D'_1 + \cdots + D'_n < D_1 + \cdots + D_n + D'' \quad (5)$$

To put (4) and (5) into chain constraint forms, we need to translate the gate delays into delays between events. This can be done in several ways, depending on whether events represent rising or falling signal transitions, thus producing chain constraints with different base states. Furthermore, one might insert extra delays in the right-hand sides to constrain events that are not consecutive; such an example was encountered in [Ne97]. This might seem complicated, but usually it is quite clear which chain constraints need to be selected, by examining the counter-example executions that are produced by the tool when checking the circuit *without* the respective constraints. Moreover, in most cases, just a few of the constituent chain constraints of each fork suffice to guarantee correctness, and we can ignore the rest of them, facilitating both the analysis and the implementation of the circuit.

One way to handle the genuine isochronic forks from [Ma90] is as depth-one extended isochronic forks, modeling the branch delays by inertial buffers.

3.6 Thresholds and Constraint Interpretations

Chain constraints were defined in Subsection 3.4 under a discrete-state assumption, but, to implement them in a circuit, one needs to translate them into constraints on the dynamical-system parameters of the circuit. Aiming for a safe implementation, we propose a translation which is always pessimistic.

Links between the discrete-state and dynamical system viewpoints have been extensively studied, for instance, in [GC94] and [KM91]; however, we need to discuss the issue from a different perspective. The intention of [GC94] is to rely solely on topological features of the phase spaces, whereas, to define delays, we associate events to crossings of voltage thresholds. In [KM91] the phase spaces are partitioned into fixed boxes corresponding to discrete states, whereas we need to allow some uncertainty for the thresholds. We sometimes associate more than one discrete execution to each analog trajectory; for example, the

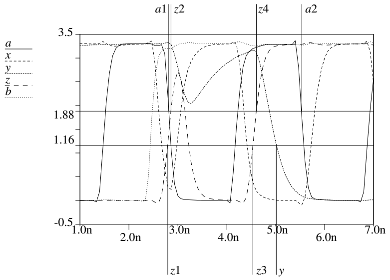


Figure 6: Meaning of delay constraint $D_{z\uparrow a\downarrow} > D_{z\uparrow y\downarrow}$, and simulation for Subsection 5.1.

cell	a	b	c
INV	1.55 V		
NOR2	1.61 V	1.73 V	
NOR3	1.60 V	1.68 V	1.78 V
NAND2	1.46 V	1.36 V	
NAND3	1.40 V	1.35 V	1.26 V

Table 1: Simulated logic thresholds, per input, for cells from a 0.5μ CMOS library, operated at 3.3 V.

waveform in Fig. 6 can be represented by either one of the executions

$$a\!\uparrow b\!\uparrow x\!\downarrow z\!\uparrow a\!\downarrow x\!\uparrow z\!\downarrow a\!\uparrow x\!\downarrow z\!\uparrow y\!\downarrow a\!\downarrow x\!\uparrow a\!\uparrow \quad \text{or}$$
$$a\!\uparrow b\!\uparrow x\!\downarrow a\!\downarrow z\!\uparrow x\!\uparrow z\!\downarrow a\!\uparrow x\!\downarrow z\!\uparrow y\!\downarrow a\!\downarrow x\!\uparrow a\!\uparrow$$

since the first $a\downarrow$ and $z\uparrow$ transitions cross their thresholds concurrently.

We consider two logical thresholds, T_{low} and T_{high}, that enclose the range of gate thresholds that are physically possible, and maybe some extra margins to account for fluctuations. Due to dependences on transistor widths and other effects, the physical thresholds for a given technology differ among gates and even among inputs of the same gate. As in [vB92], we define physical thresholds by matching saturation currents through the pull-down and pull-up networks of a cell. Some other definitions of physical thresholds can also be accommodated in our method, as long as they allow for the variation in threshold values. (However, a definition based on unit gain points would make the threshold ranges unnecessarily wide.) The threshold values determined by our simulations are given in Table 1. To filter out delay effects, we have used slow variations of the input signals. In this paper we use the range 1.16–1.88 V, which covers these values plus and minus a margin of 0.1 V, chosen arbitrarily.

The purpose of a chain constraint of the form (1) is to ensure that event (signal transition) a_m occurs before b_n. We take this to mean that the signal of a_m *leaves* the logical threshold range before the signal of b_n *enters* the range.

The meaning of a chain constraint $D_{z\uparrow a\downarrow} > D_{z\uparrow y\downarrow}$ is illustrated in Fig. 6. This constraint is first interpreted as $D_{z\,a_1} > D_{z\,y}$, where z can be a point on the time axis situated between z_1 and z_2. In this first interpretation, the constraint is clearly violated. A second interpretation is $D_{z'a_2} > D_{z'y}$, where z' is a point between z_3 and z_4; here, the constraint is obeyed.

A consequence of the proposed interpretations for chain constraints is that, if a wider threshold range is used, the chain constraints become stricter and harder to implement. This observation agrees with the concern expressed in [vBHP95] and [vB92] for reducing the possible variation of the thresholds. In our method, however, threshold spreads are not crucial. If the chain constraints themselves are not very tight (if their safety margin is high—see Subsection 3.7) then we might afford large threshold spreads.

Further research is necessary to develop a practical method to implement chain constraints in a given technology. One approach is to size each of the delays $D_{a_i a_{i+1}}$ and $D_{b_j b_{j+1}}$ in a constraint like (1), imposing lower and upper bounds on each delay individually, so that the sum of the lower bounds on $D_{b_j b_{j+1}}$ is always greater than the sum of the upper bounds on $D_{a_i a_{i+1}}$. Because of the differences between upper and lower bounds, chain constraints may become harder to implement if they involve longer chains.

3.7 Safety Margins

To estimate how hard it is to implement chain constraints, one can use the safety margin measure from [KN94], which evaluates the timing reliability of a circuit in the presence of delay fluctuations. Given a constraint of the form (1), its safety margin can be defined as the ratio i/j, where i and j are the numbers of inversions on the greater and smaller sides of the inequality sign, respectively. Note that we count CMOS inversions rather than gates. A larger ratio grants more leeway for the delays involved, presumably making the constraints easier to implement.

As in [KN94], the safety margin of a set of chain constraints is the *smallest* of the safety margins of the chain constraints in the set. In general, a circuit has multiple sets of chain constraints that each guarantee its correctness. The safety margin of a circuit can be defined as the *largest* of the safety margins of these sets.

The number of CMOS inversions is an oversimplification of the delay of a gate, although it does serve as a first-order approximation. More detailed models can be used as well, for instance models that take the capacitive load into account, and thereby model both the fanout and the wiring of a circuit. Delay constraints then translate into bounds on the capacitances. This, however, is not further pursued in this paper.

4 Verification Strategy

The verification strategy that is followed in this paper, and more precisely, in the next section, is illustrated in Fig. 7. We repeatedly check refinement between a specification process and the product of CMOS cell processes and constraint processes. If refinement holds, the circuit behavior is found to be correct under the current cell models and assumed constraints. If refinement does not hold, we obtain a 'witness to failure' in the form of an execution that contradicts the refinement relationship. Often, the detected 'failures' do not indicate actual flaws in the circuit functionality, but just hazards that can be tolerated, or timing dangers that can be avoided by additional delay constraints. In the case of a tolerated hazard, the hazard-intolerant cell models that are used by default need to be replaced by other models that do not declare the respective hazard to be a reject. In the case of a timing problem, the counter-example execution typically contains certain events out of order, calling for a new chain constraint to put the respective events in the desired order. The verification is resumed using adjusted cell models and additional delay constraints, until either refinement holds or a functional flaw is found.

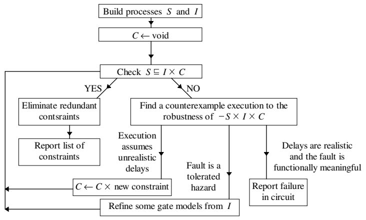


Figure 7: The verification stategy followed in this paper.

The case studies presented in this paper are rather simple, enabling us to readily verify the constraints involved. More complex examples (approximately 10^{11} reached states) have also been verified using this strategy [Ne98].

Chain constraints can increase the complexity of the state spaces of the systems where they are used, compounding the state-explosion problem. This problem can be alleviated as usual, by the use of BDDs and by modular and hierarchical verification. (Since the algebraic properties of process spaces that justify structured verification have no restrictions, they do apply, in particular, to systems that use chain constraints.) In addition, there are techniques specific to chain constraints that can also reduce the state spaces, by sharing states between constraints that have several circuit signals in common.

5 Case Studies

In [Pe96], the implementation of handshake circuits [vB93] in a generic standard-cell library —containing no dedicated asynchronous cells like C-elements— is described. The time dependences in these circuit implementations generally take the form of chain constraints. The analysis technique described above is applied to specify these timing assumptions precisely and to verify them.

5.1 C-Element with Delay Constraints

Fig. 8 shows a circuit from [Pe96] that implements a three-input C-element in a speed-dependent manner. The '==' sign denotes an extended isochronic fork of depth one, as defined in [vBHP95], and an extra gate with input z is assumed to intervene in the environment delay from z to a, b, and c.

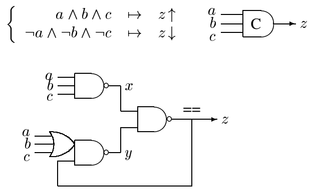


Figure 8: Three-input C-element: production rules, symbol, and implementation [Pe96, p. 70].

First, we check whether the circuit satisfies its specification in the absence of delay assumptions. The specification is the hazard-intolerant model of a three-input C-element. The circuit is modeled as the product of hazard-intolerant models of the circuit gates, with initial state a,

b, c, and z low and x and y high. To check whether the circuit satisfies the specification, we verify by FIREMAPS the refinement relationship between the specification process and the circuit model. The answer is that refinement does not hold, and the counter-example that is produced is execution $c\uparrow b\uparrow a\uparrow x\downarrow z\uparrow a\downarrow a\uparrow y\downarrow$. This execution is a reject for the NAND having output x, and a goal for all other gate processes and the specification process. The smallest reject prefix is $c\uparrow b\uparrow a\uparrow x\downarrow z\uparrow a\downarrow a\uparrow$, indicating that a hazard may occur at the NAND, although it is not a hazard for the specification. To see whether this hazard may produce a fault that is externally visible, we replace the circuit gate models by inertial models, and we obtain $c\uparrow b\uparrow a\uparrow x\downarrow z\uparrow a\downarrow x\uparrow z\downarrow$, which is an error for the specification and a goal for the circuit gates. This execution contains an illegal $z\downarrow$ transition; a similar execution was obtained by simulation in Fig. 6.

To avoid the danger above, [Pe96] uses the delay assumption of an extended isochronic fork (denoted by '==' in Fig. 8). To avoid the illegal $z\downarrow$ in $c\uparrow b\uparrow a\uparrow x\downarrow z\uparrow a\downarrow x\uparrow z\downarrow$, we only need to ensure that after $z\uparrow$ the internal signal y has enough time to stabilize before the next input transition. Thus we only impose the following chain constraints, which derive from the fork assumption for the delay introduced by the extra environment gate mentioned above:

$$D_{z\uparrow a\downarrow} > D_{z\uparrow y\downarrow} \quad (6)$$
$$D_{z\uparrow b\downarrow} > D_{z\uparrow y\downarrow} \quad (7)$$
$$D_{z\uparrow c\downarrow} > D_{z\uparrow y\downarrow} \quad (8)$$

Chain constraint (6) is illustrated in Fig. 5 and Fig. 6. This chain constraint guarantees avoidance of the execution $cbaxzaxz$ above by making it an error of its process. (Actions other than a, y, and z do not change the state of the chain constraint process and are omitted from Fig. 5.)

Still, the product of chain constraint processes and the hazard-intolerant processes of the circuit gates does not refine the specification, a counter-example execution being $c\uparrow b\uparrow a\uparrow x\downarrow z\uparrow y\downarrow a\downarrow a\uparrow$. This execution shows a hazard by $a\uparrow$ at the NAND gate that drives x, although this behavior is legal for the environment. We need to determine whether this hazard can lead to a 'true' fault (externally visible), or whether it can be ignored.

Under an inertial model for this NAND gate and hazard-intolerant models for all other gates, the circuit with the chain constraints above does satisfy the hazard-intolerant three-input C-element specification. But is an inertial model appropriate for this gate?

Since the NAND gate that drives x is an inverting gate and its load is not abnormally small, one may expect that an inertial model will be appropriate for this gate. The verification procedure can compare exhaustively the discrete-state models of the components and the specification, but other methods have to be employed to determine whether the models used describe properly the actual, physical behavior of the system. To validate the model for this NAND gate, we have simulated negative pulses of varying durations on input signal a, while keeping b and c high. We have made the input transitions abrupt, to make it easier for the output pulse to be delayed beyond the input pulse and expose non-inertial behavior, if any. The gate parameters and the load had typical values currently used for this circuit. (This load was the capacitance of an internal node.) The response of the gate was indeed inertial: either the output pulse overlapped the input pulse, as in Fig. 9 (a), or the output spike was too small to be perceived as a pulse by other gates, as in Fig. 9 (b).

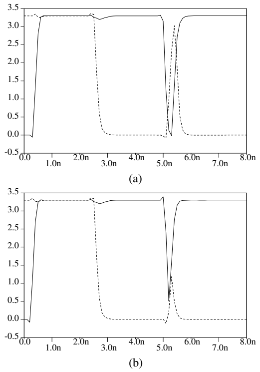


Figure 9: Response to an input pulse, for a NAND gate with output x (dotted line) and negative pulses of varying lengths on input a (solid line): (a) an output pulse is produced, but it overlaps the input pulse; (b) the output spike becomes too small.

5.2 Multiplexer Control

Fig. 10 shows a multiplexer component taken from [vBHP95, Pe96]. The environment of this component is assumed to guarantee mutual exclusion between handshakes on channels a and b. The component will forward incoming data from a or b along c and, when acknowledged through c, send an acknowledge along a or b, depending on the origin of the data. The component thus multiplexes data from a and b on channel c.

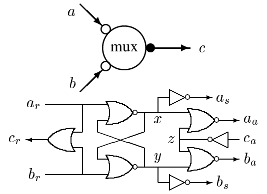


Figure 10: Multiplexer component and control circuit [Pe96, p. 96].

In the single-rail implementation of this component, data and control are treated separately. A control circuit (shown in Fig. 10) connects to the control wires of each handshake channel (denoted by subscripts r for request and a for acknowledge) and from this generates select wires (denoted by subscript s). The select wires connect to the data part of the circuit, which for each bit i implements function $c_i = a_i \wedge a_s \vee b_i \wedge b_s$. The data on output c should be valid at least from $c_r\downarrow$ until $c_a\downarrow$, given that the input also guarantees this so-called 'late' data-valid scheme [Pe96]. During a handshake on a, for instance, data is stable at least from $a_r\downarrow$ until $a_a\downarrow$.

The specification of the multiplexer control circuit is given by the following command:

$$\begin{array}{l} *(\, b_r\uparrow ;\ (a_s\downarrow \,\|\, b_s\uparrow \,\|\, c_r\uparrow ;\ c_a\uparrow ;\ b_a\uparrow ;\ b_r\downarrow) ; \\ \quad c_r\downarrow ;\ c_a\downarrow ;\ b_a\downarrow \\ \;\, |\, a_r\uparrow ;\ (a_s\uparrow \,\|\, b_s\downarrow \,\|\, c_r\uparrow ;\ c_a\uparrow ;\ a_a\uparrow ;\ a_r\downarrow) ; \\ \quad c_r\downarrow ;\ c_a\downarrow ;\ a_a\downarrow \\ \;\,) \end{array}$$

Operator '$*$' is used for repetition, ';' for concatenation, '$\|$' for parallel composition, and '$|$' for choice. They are listed here in decreasing order of precedence.

Transitions on a_s and b_s are conditional, that is, they will be ignored if their corresponding signals already have the targeted levels. For instance, $a_s\uparrow a_s\uparrow$ is legal, but the second $a_s\uparrow$ has no effect. In the verification we used a more complex but equivalent command, with unconditional transitions, in which an initial state for these signals is assumed. We assumed that signals y, z, and a_s start high and all the other signals start low. The analysis would be symmetrical for an initial state where only x, z, and b_s are high.

First, we check whether the circuit satisfies the specification under the following delay assumptions indicated in [Pe96], and hazard-intolerant models for all cells:

$$D_{a_r\uparrow c_a\uparrow z\downarrow} > D_{a_r\uparrow x\downarrow a_s\uparrow} \quad (9)$$

$$D_{b_r\uparrow c_r\uparrow c_a\uparrow z\downarrow} > D_{b_r\uparrow y\downarrow b_s\uparrow} \quad (10)$$

and we obtain an execution

$$b_r\uparrow c_r\uparrow c_a\uparrow y\downarrow b_s\uparrow z\downarrow a_a\uparrow a_r\uparrow b_a\uparrow$$

that contravenes to the specification because $a_a\uparrow$ inadvertently occurs in response to $b_r\uparrow$. (Notice that $a_r\uparrow$ does not constitute a violation of the mutual exclusion requirement by the environment, since the device faulted first by issuing $a_a\uparrow$.)

To avoid this danger we need to force an $x\uparrow$ event to occur before $z\downarrow$. Thus we introduce new constraints for $a_s\downarrow$ and $b_s\downarrow$ similar to those for $a_s\uparrow$ and $b_s\uparrow$:

$$D_{b_r\uparrow c_r\uparrow c_a\uparrow z\downarrow} > D_{b_r\uparrow y\downarrow x\uparrow a_s\downarrow} \quad (11)$$
$$D_{a_r\uparrow c_r\uparrow c_a\uparrow z\downarrow} > D_{a_r\uparrow x\downarrow y\uparrow b_s\downarrow} \quad (12)$$

With chain constraints (9), (10), (11), and (12), the circuit satisfies the specification.

Let us compute the safety margin for constraints (9)–(12). Assuming $D_{c_r\uparrow c_a\uparrow}$ is larger than two inversion delays, we obtain a safety margin of $5/3$. We have not verified the datapath, but it may add two extra inversion delays to the delay of the inverters that control a_s and b_s. With these extra delays, the safety margin becomes $5/5$. If we also worry that $D_{c_r\uparrow c_a\uparrow}$ might be smaller than two inversion delays, which could happen for instance if there are extended isochronic forks in the environment, the safety margin is $3/5$, which is unacceptably small.

Fortunately, the constraints above could be split up and relaxed, by binding only x and y to $z\downarrow$, and a_s and b_s only to $c_r\downarrow$. This results in the following set of constraints:

$$D_{a_r\uparrow c_r\uparrow c_a\uparrow z\downarrow a_a\uparrow a_r\downarrow c_r\downarrow} > D_{a_r\uparrow x\downarrow a_s\uparrow} \quad (13)$$
$$D_{b_r\uparrow c_r\uparrow c_a\uparrow z\downarrow b_a\uparrow b_r\downarrow c_r\downarrow} > D_{b_r\uparrow y\downarrow b_s\uparrow} \quad (14)$$
$$D_{a_r\uparrow c_r\uparrow c_a\uparrow z\downarrow a_a\uparrow a_r\downarrow c_r\downarrow} > D_{a_r\uparrow x\downarrow y\uparrow b_s\downarrow} \quad (15)$$
$$D_{b_r\uparrow c_r\uparrow c_a\uparrow z\downarrow b_a\uparrow b_r\downarrow c_r\downarrow} > D_{b_r\uparrow y\downarrow x\uparrow a_s\downarrow} \quad (16)$$
$$D_{a_r\uparrow c_r\uparrow c_a\uparrow z\downarrow} > D_{a_r\uparrow x\downarrow y\uparrow} \quad (17)$$
$$D_{b_r\uparrow c_r\uparrow c_a\uparrow z\downarrow} > D_{b_r\uparrow y\downarrow x\uparrow} \quad (18)$$

With the relaxed constraints (13)–(18), the circuit also satisfies the specification.

Assuming two inversion delays in the datapath and in $D_{c_r\uparrow c_a\uparrow}$, and one inversion delay in $D_{a_a\uparrow a_r\downarrow}$, the safety margins are $9/4$ for (13) and (14), $9/5$ for (15) and (16), and $5/2$ for (17) and (18). These constraints are thus very weak and should be easy to implement. Even if no environment delay can be reliably assumed and if there are additional datapath inversion delays, the safety margins are $6/4$ for (13) and (14), $6/5$ for (15) and (16), and $3/2$ for (17) and (18), which are still quite good.

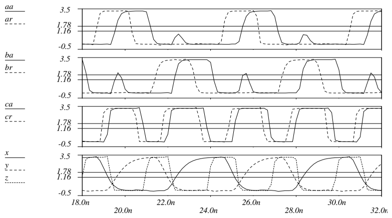


Figure 11: Simulation of the failure mode of execution (19).

None of chain constraints (13)–(18) is made redundant by the others. If only some of these constraints are used, we obtain violations of the specification. Using only (15)–(18), $c_r\!\downarrow$ comes before $b_s\!\uparrow$ in

$$b_r\!\uparrow c_r\!\uparrow c_a\!\uparrow y\!\downarrow x\!\uparrow z\!\downarrow b_a\!\uparrow a_s\!\downarrow b_r\!\downarrow c_r\!\downarrow b_s\!\uparrow .$$

Using only (13), (14), (17), and (18), $c_r\downarrow$ comes before $a_s\!\downarrow$ in

$$b_r\!\uparrow c_r\!\uparrow c_a\!\uparrow y\!\downarrow x\!\uparrow z\!\downarrow b_a\!\uparrow b_s\!\uparrow b_r\!\downarrow c_r\!\downarrow a_s\!\downarrow .$$

Using only (13)–(16), $a_a\!\uparrow$ inadvertently responds to $b_r\!\uparrow$ in

$$b_r\!\uparrow c_r\!\uparrow c_a\!\uparrow z\!\downarrow a_a\!\uparrow a_r\!\uparrow b_r\!\downarrow . \qquad (19)$$

To show the importance of these constraints, we have simulated the failure mode of execution (19) by adding extra load to the cross-coupled NOR gates, to increase the delays of these gates to a point where chain constraints (17) and (18) are violated. This simulation is shown in Fig. 11. Constraint (17) would impose that $z\!\downarrow$ enters the threshold range after $y\!\uparrow$ leaves the range, which does not happen for the simulation in Fig. 11. If the constraint were complied to, the erroneous pulses would not appear on b_r.

5.3 DO-Component

The DO-component, which is shown in Fig. 12 (a), is used to implement repetition [vB93]. When activated along passive handshake port a, it evaluates the guard connected to b, and depending on the outcome, either performs a handshake along c and evaluates again, or completes the handshake along a. A precise specification of the four-phase behavior is in Fig. 13.

Further we note that the b_0 guard can only be changed after the handshake on a has been completed, thus b_0 will be stable at least till $a_a\!\downarrow$. The state graph of the specification, shown in Fig. 13, has two main cycles, one for b_0 low and one for b_0 high, which communicate via edges labeled b_0. Notice that signal b_0 is supposed to be constant from $b_a\!\downarrow$ till $a_a\!\downarrow$ or $c_r\!\uparrow$.

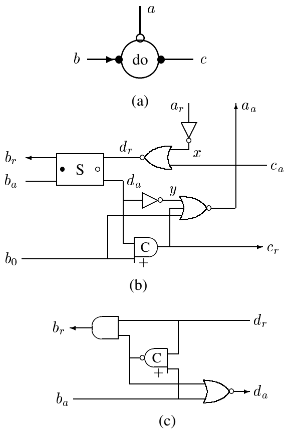


Figure 12: DO-component: (a) symbol; (b) circuit [Pe96]; (c) S-element implementation.

The symbol of the DO-component is shown in Fig. 12 (a). A possible gate realization of the DO component (taken from [Pe96, p. 79]) is shown in Fig. 12 (b). The component marked 'C' is a generalized C-element [Ma90], given by the following production rules:

$$\begin{cases} b_0 \wedge d_a & \rightarrow & c_r\!\uparrow \\ \neg d_a & \rightarrow & c_r\!\downarrow \end{cases}$$

The box marked 'S' is an S-element[2], whose implementation is shown in Fig. 12 (c). The S-element is specified by the following command ([Pe96, p. 75]):

$$*(d_r\!\uparrow b_r\!\uparrow b_a\!\uparrow b_r\!\downarrow b_a\!\downarrow d_a\!\uparrow d_r\!\downarrow d_a\!\downarrow)$$

To start, we have verified that the S-element specification is refined by the product of hazard-intolerant models of the gates in Fig. 12 (c).

In the analysis, we assume the circuit in 12 (b) starts at a state where signals x and y are high and all other signals are low. If all inputs are held low, the circuit does indeed stabilize at this state.

[2] The S-element is also called 'Q' in [Ma90]

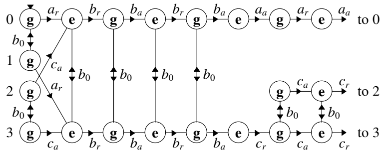


Figure 13: State graph for the DO-component specification.

In [Pe96], it was also assumed that the delay from a $d_a\!\downarrow$ event to a $c_r\!\downarrow$ event through the C-element (two inversions) is larger than the delay from $d_a\downarrow$ to $y\uparrow$ (one inversion), which translates into the chain constraint

$$D_{d_a\downarrow c_r\downarrow} > D_{d_a\downarrow y\uparrow} . \qquad (20)$$

With constraint (20) and hazard-intolerant models of all gates, we obtained the following counter-example to refinement, indicating a hazard by $d_a\!\downarrow$ at the inverter whose output is y:

$$a_r\!\uparrow x\!\downarrow d_r\!\uparrow b_r\!\uparrow b_0\!\uparrow b_a\!\uparrow b_r\!\downarrow b_a\!\downarrow d_a\!\uparrow c_r\!\uparrow c_a\!\uparrow d_r\!\downarrow d_a\!\downarrow c_r\!\downarrow$$

Is this hazard a true danger of violating the specification? Using an inertial model for the inverter and hazard-intolerant models for all other gates, the circuit with constraint (20) does refine the DO-component specification above.

Better yet, we do not need to rely on an inertial model for the inverter. With constraint (20) and the additional constraint

$$D_{d_a\uparrow c_r\uparrow} > D_{d_a\uparrow y\downarrow} \qquad (21)$$

no hazard occurs: under hazard-intolerant models for all gates, the circuit with constraints (20) and (21) refines the DO-component specification above. The circuit works fine under the (rather weak) chain constraints (20) and (21), which imply a safety margin of at least $2/1$.

6 Conclusion

We have verified three circuits from [Pe96] as case studies for the analysis technique of [Ne97]. This exercise has pointed to several problems at the border between discrete and analog viewpoints on circuit behavior. We have given an account of these problems and the way we dealt with them. The circuits were found to be correct, and in one case a notable relaxation of the delay constraints could be obtained. The three analysis sessions in Section 5 took 7, 48, and 8 seconds on a Sun Sparc 5/110 to perform 4, 6, and 4 verifications or diagnoses, respectively, including the delay constraints. Also, we have performed analog simulations for the circuits under study, and the simulations have been consistent with the discrete-state analysis.

Maybe we could have performed our analysis using a different framework. Process spaces are closely related and owe to several 'trace-oriented' treatments of concurrency, especially to those in [Di89], [Eb91], [Ho85], [Jo92], and [Ve94]. However, in their present forms, [Di89], [Jo92], and [Ve94] forbid connecting outputs by parallel composition, whereas in our analysis we need to take the product of chain constraint processes and gate processes that share control over certain actions, and we need the structured verification properties to cover this connectivity case too. CSP and [Eb91] do allow shared outputs in parallel composition, but they have, for instance, restrictions on the processes that can participate in comparisons.

Previous timing analysis methods from [La92] or [My95] allow for a more detailed analysis than we do here, but they require numerical bounds on individual delays, and in our examples such bounds are not available.

Because chain constraints provide for the cases where the involved signal transitions occur out of order or do not occur at all, we did not treat them as simple linear inequalities on delays. On the other hand, chain constraints are easy to handle as processes, and as such they can deal with deadlock, hazards, and non-determinism. The safety margin measure from [KN94] could also be applied for chain constraints.

An interesting direction for further work may be an exhaustive study of the behavior of CMOS cells in the presence of hazards, comparing analog simulations to the discrete-state models that are used by the verification procedure. Once the behavior in the presence of hazards is properly modeled, one might reduce the use of hazard-intolerant models in verification, focusing directly on the externally visible, non-hazard faults.

Acknowledgments We are grateful to C.H. van Berkel, J.A. Brzozowski, and J.L.W. Kessels for comments on the manuscript. Also, C.E. Molnar's concern for links between our discrete-state analysis and lower levels of circuit modeling, expressed in [MJCL97], is gratefully acknowledged.

References

[vB92] K. van Berkel. Beware the isochronic fork. *Integration, the VLSI journal*, 13(2):103–128, June 1992.

[vB93] K. van Berkel. *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*, volume 5 of *International Series on Parallel Computation*. Cambridge University Press, 1993.

[vBHP95] K. van Berkel, F. Huberts, and A. Peeters. Stretching quasi delay insensitivity by means of extended isochronic forks. In *Asynchronous Design Methodologies*, pages 99–106. IEEE Computer Society Press, May 1995.

[BRB90] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, June 1990.

[BS95] J. A. Brzozowski and C.-J. H. Seger. *Asynchronous Circuits*. Springer-Verlag, 1995.

[Di89] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1989.

[Eb91] J. C. Ebergen. A formal approach to designing delay-insensitive circuits. *Distributed Computing*, 5(3):107–119, 1991.

[GC94] M. R. Greenstreet and P. Cahoon. How fast will the flip flop? In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 77–86, 1994.

[Ho85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Jo92] M. B. Josephs. Receptive process theory. *Acta Informatica*, 29(1):17–31, 1992.

[KM91] R. Kurshan and K. McMillan. Analysis of digital circuits through symbolic reduction. *IEEE Transactions on Computer-Aided Design*, 10(11):1356–1371, 1991.

[KN94] M. Kuwako and T. Nanya. Timing-reliability evaluation of asynchronous circuits based on different delay models. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 22–31, November 1994.

[La92] L. Lavagno. *Synthesis and Testing of Bounded Wire Delay Asynchronous Circuits from Signal Transition Graphs*. PhD thesis, U.C. Berkeley, November 1992.

[Ma90] A. J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication*, UT Year of Programming Series, pages 1–64. Addison-Wesley, 1990.

[MJCL97] C. E. Molnar, I. W. Jones, B. Coates, and J. Lexau. A FIFO ring oscillator performance experiment. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, April 1997.

[My95] C. J. Myers. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*. PhD thesis, Dept. of Elec. Eng., Stanford University, October 1995.

[Ne95] R. Negulescu. Process spaces. Technical Report CS-95-48, Computer Science, University of Waterloo, December 1995. `http://maveric.uwaterloo.ca/~radu/ps.html`

[Ne97] R. Negulescu. A technique for finding and verifying speed-dependences in gate circuits. In *ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Austin, Texas, December 1997. A technical report version is available from `http://maveric.uwaterloo.ca/~radu/ps.html`

[Ne98] R. Negulescu. Event-driven verification of switch-level correctness concerns. In *International Conference on Application of Concurrency to System Design*, Aizu-Wakamatsu, Japan, March 1998.

[Pe96] A. M. G. Peeters. *Single-Rail Handshake Circuits*. PhD thesis, Eindhoven Univ. of Technology, Eindhoven, The Netherlands, June 1996.

[Ve94] T. Verhoeff. *A Theory of Delay-Insensitive Systems*. PhD thesis, Eindhoven Univ. of Technology, Eindhoven, The Netherlands, May 1994.