Delay-Insensitivity and Ternary Simulation*

J. A. BRZOZOWSKI
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
email: brzozo@uwaterloo.ca
March 1997

Abstract. Consider a network N constructed from a set of modules interconnected by wires. Suppose that there is a formal specification Σ for N, and that the behavior of N satisfies this specification. Let $\hat{N}$ consist of the same modules, but suppose that these modules and the interconnecting wires have arbitrary delays. We say that N is delay-insensitive if the behavior of any network $\hat{N}$ defined as above still satisfies Σ. An important problem in asynchronous circuits is to determine, given a specification Σ and a set $\mathcal{T}$ of module types, whether there exists a delay-insensitive network of modules from $\mathcal{T}$ with a behavior satisfying Σ. If such a network exists, we say that it implements Σ. In the case where the components are logic gates, it is known that very few specifications have delay-insensitive implementations. The proofs of several such results involve "ternary simulation"—an analysis method based on ternary algebra— and rely on a key theorem linking binary analysis and ternary simulation. In this paper we survey the known results concerning delay-insensitivity, and outline one proof that a simple specification cannot be implemented.

Keywords: asynchronous, circuit, delay-insensitive, fundamental mode, general multiple-winner, input/output mode, network, semi-modular, speed-independent, ternary algebra, ternary simulation.

1 Introduction

Although most currently used digital circuits are synchronous, that is, operate under the control of a clock signal, there is considerable interest in asynchronous circuits [7]. Clock-free design offers some advantages, like the

*This research was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871.

potential for lower energy consumption, higher speed, and avoidance of clock distribution problems.

Among asynchronous designs, the class of so-called delay-insensitive networks is receiving special attention. Roughly speaking, a network is delay-insensitive if it continues to operate correctly, even if the delays in its components and wires change arbitrarily. When such networks are designed in a modular fashion, it is possible to replace their components by faster or slower ones, without changing the correctness of the network operation, although, of course, performance may be affected.

The following discussion attempts to treat delay-insensitivity in a very general fashion. For this reason, the concepts are not formally defined, but we appeal to the reader's intuition. More precise statements are presented later. We have in mind the following concepts. A module is some kind of basic element like a gate or an arbiter. A network consists of a set of modules interconnected by wires in some fashion. With each network we associate its behavior. For example, a behavior may be specified by a trace structure (a set of sequences of possible actions), or by a finite automaton, or by a directed graph. We also assume that there is a notion of "implementation" of one behavior by another. Now suppose a network N is constructed from a set of modules, and that its behavior implements some specification Σ. Let $\hat{N}$ consist of the same modules and interconnecting wires, but suppose that these modules and wires have arbitrary delays. We say that N is *delay-insensitive* if the behavior of any network $\hat{N}$ defined as above still implements Σ.

An important problem in asynchronous circuits is to determine, given a specification Σ of an asynchronous behavior and a set $\mathcal{T}$ of module types, whether there exists a delay-insensitive network of modules from $\mathcal{T}$ that implements Σ. In the case where the components are logic gates, it is known that very few specifications have delay-insensitive implementations. In this paper we state several such results; their proofs involve "ternary simulation"—an analysis based on ternary algebra— and rely on a key theorem linking binary analysis and ternary simulation. To illustrate the applicability of ternary simulation, we discuss a proof that a very simple behavior has no delay-insensitive implementation by a gate circuit.

The paper is structured as follows. In Section 2 we briefly survey the previous results concerning delay-insensitivity, concentrating on the various definitions of delay-insensitivity. We then define our model of gate networks in Section 3, and describe the binary model of network behavior in Section 4. The ternary analysis method known as "ternary simulation" is next

presented in Section 5, and the main results relating ternary simulation and binary analysis are summarized. The limitations of gate networks operated in fundamental mode are discussed in Section 6, and Section 7 treats circuits operated in input/output mode. Section 8 concludes the paper.

2 Delay-Insensitivity

We now briefly survey the literature on delay-insensitivity, concentrating on the formal definitions of the concept, rather than on design methods resulting in delay-insensitive circuits. Most of these definitions involve such concepts as "a circuit B satisfies a specification A," or "a circuit with added delays (B) behaves like the original circuit A," or "a circuit B implements a behavior A." Because of space limitations, we are unable to make these concepts much more precise. We point out, however, that many different approaches have been taken in the past to formalize these concepts. The formalizations range from requiring that "A and B go to the same final stable state," to "A and B have the same trace structures," to "A and B have the same trace structures and deadlock is not possible in B," etc. Because of this diversity, we refer the interested reader to the original sources.

A very early (1955-59) work concerned with delays in circuits is that of D. Muller on "speed-independent" circuits. This work was described in a number of technical reports from the University of Illinois (see pp. 242–243 in [24]), but is more easily available in the book by R. E. Miller [24]. Muller considered *autonomous* circuits, i.e., circuits without external inputs. He assumed that components could have arbitrary delays, but that wire delays were negligible. In Muller's approach, the behavior of a circuit is described by *allowed sequences* of states, which specify the order in which the state variables may change, but not the times of change. Only changes of state are recorded; thus, no two consecutive states in an allowed sequence are equal. An allowed sequence may be finite—in which case it must end in a state which has no successors—or infinite. Any given state may have a number of possible successor states; the choice of successor depends on the relative sizes of circuit delays.

Two states b and c of a circuit are *equivalent* if there is an allowed sequence in which b follows c and also c follows b. The relation "follows" induces a partial order on the equivalence classes of a circuit. Consequently, every allowed sequence has a unique *terminal* class. A circuit is said to be *speed-independent with respect to a state* q, if all allowed sequences starting

with q have the same terminal class. In case the terminal classes consist of single states, this condition implies that, starting in q, the network can only end up in a unique stable state. Otherwise, if the terminal class has more than one state, then it corresponds to a cycle of states, and all the paths starting from q must end up in the same cycle. Because of differences in component delays, there may be many different paths from q. If the circuit is speed-independent, however, the final destination (state or cycle) of any path from a given state is uniquely determined by that state.

Muller defined another important concept related to proper circuit operation, namely the property of semi-modularity. A circuit is *semi-modular with respect to state* q, if the following condition is satisfied by every state b reachable from q: No transition from one state to another can affect the excitation of any unstable variable that does not change during that transition. More precisely, if b is a state in which s_i is unstable and c is an immediate successor state of b, then either s_i changes to its excitation or s_i is still unstable in state c. Semi-modular circuits are a proper subclass of speed-independent circuits, that is, every circuit that is semi-modular with respect to q is also speed-independent with respect to q, but the converse is false.

Although the term "delay-insensitive" had not yet been introduced, an early (1959) result due to Unger deals with limitations of delay-insensitive circuits [37, 38]. Unger has shown that an asynchronous behavior that contains a so-called essential hazard is not implementable by a gate network with arbitrary gate and wire delays. We return to this result in Section 6.

The term *delay-insensitive circuit* originated from the Macromodules project, which was carried out at Washington University in St. Louis, Missouri around 1970 (see [26] for a short description of the project). This term was used informally. C. E. Molnar introduced the "foam-rubber wrapper" postulate to describe delay-insensitive specifications of network modules [26]. A module has a number of input terminals on which it receives signals from its environment, and a number of output terminals on which it sends signals to the environment. Because of wire delays, however, signals produced by the environment are not seen by the module right away, but only after some delay. Similarly, signals produced on the module outputs are observed by the environment only after some delay. The interface delays can be represented as a "foam-rubber wrapper" surrounding the module. The inner surface of the wrapper corresponds to the module interface, whereas the outer surface defines the environment interface. The foam-rubber analogy suggests that the distance between the inner and outer surfaces along one

wire may be different from that along a different wire, representing different (and possibly time-varying) delays.

The communications between a component and its environment are specified by "traces," which are sequences of events at the input and output terminals of the component. Formally, a *trace structure* [13, 34, 40] T is a triple $T = \langle \mathbf{i}T, \mathbf{o}T, \mathbf{t}T \rangle$, where $\mathbf{i}T$ is the *input alphabet*, $\mathbf{o}T$ is the *output alphabet*, and $\mathbf{t}T$ is the *trace set*. The input and output alphabets are two disjoint finite sets corresponding to the input and output terminals of the module. An element x of $\mathbf{i}T$ (respectively, an element z of $\mathbf{o}T$) is interpreted as a signal at the input terminal x (respectively, output terminal z). The set $\mathbf{a}T = \mathbf{i}T \cup \mathbf{o}T$ is called the *alphabet* of T. A word w in $(\mathbf{a}T)^*$ is a *trace*. Since a trace represents a history of possible signals, any prefix of a trace must itself be a trace. Thus, $\mathbf{t}T$ is *prefix-closed* in this sense, and always contains the empty word ϵ that represents the initial state in which no signals have yet been sent. In trace terms, the foam-rubber wrapper postulate requires that the set of traces of a module/environment specification at the inner surface of the wrapper must be the same as the set of traces at the outer surface. For example, suppose a specification (at the inner surface) requires that a module produce output z_1 and then output z_2, in that order. Trace $z_1 z_2$ would then be in its trace set, but $z_2 z_1$ would not be in the set. Because we assume that the delays in the output wires are unknown and arbitrary, z_2 might appear before z_1 at the outer surface, or the correct order might be preserved. Thus, the trace set at the outer surface contains both $z_1 z_2$ and $z_2 z_1$. Since the inner and outer trace sets differ, the module specification is not delay-insensitive.

An important contribution to the design of asynchronous circuits is the work of C. L. Seitz (1980) on "self-timed" systems (Chapter 7 in [23]). This work contains an informal description of a design methodology of a class of asynchronous circuits. A *self-timed* system consists of modules performing computations. A computation by a module is initiated by signals at its inputs, and its completion is indicated by signals at its outputs. The modules are assumed to occupy a small enough area on a VLSI chip to justify the assumption that wire delays are negligible. Such areas are called *equipotential regions*. Thus, the modules themselves are delay-sensitive. A self-timed network of such modules, however, should operate correctly even in the presence of arbitrary wire delays. Inside an equipotential region the designer, knowing some bounds on the various delays, may cause one signal to arrive before another, and this order of signals will be preserved. Such order assumptions cannot be made about communication protocols among

equipotential regions. The system must be so designed that a module will not be called upon to initiate a new computation until it has completed its present computation, and has signalled this completion to its environment.

The first formal definition of delay-insensitivity was made by J. T. Udding in 1984 [34, 36]. Udding suggests that a delay-insensitive specification should satisfy the following two conditions. First, there should be no *computation interference,* meaning informally that no signal should be sent to a module if that module is not ready to receive it. Similarly, the module should not send a signal to the environment, if the latter is not ready to receive it. Second, there should be no *transmission interference,* meaning informally that two consecutive signals cannot be sent along any wire. Udding then defines a trace structure $T = \langle \mathbf{i}T, \mathbf{o}T, \mathbf{t}T \rangle$ of a module to be delay-insensitive if it satisfies the four rules stated below. Two symbols a and b in $\mathbf{a}T$ are said to be of the *same type* if they are both inputs or both outputs; otherwise, they are of *different types*. The rules are:

1. There should never be two consecutive signals along the same wire. Formally, for any $w \in \mathbf{t}T$ and $a \in \mathbf{a}T$, $waa \notin \mathbf{t}T$.

2. There should not be any ordering between input signals of a module, and the same holds for output signals. Formally, for any $u, v \in \mathbf{t}T$, and a and b of the same type, $uabv \in \mathbf{t}T$ if and only if $ubav \in \mathbf{t}T$.

3. Suppose signals a and b are of different types, and c is of the same type as a. If a and b can occur in either order at some point, and if c can occur when the order is ab, then it can also occur when the order is ba. Formally, for any $u, v \in \mathbf{t}T$, if $uabv \in \mathbf{t}T$ and $ubav \in \mathbf{t}T$, then $uabvc \in \mathbf{t}T$ implies $ubavc \in \mathbf{t}T$.

4. If a and b are signals of different types, and both are enabled to occur after some trace w, then the occurence of one signal should not disable the occurrence of the other signal. Formally, for any $w \in \mathbf{t}T$ and a and b of different types, if $wa \in \mathbf{t}T$ and $wb \in \mathbf{t}T$, then $wab \in \mathbf{t}T$.

Udding has shown that, if a delay-insensitive trace structure and its environment satisfy the four rules above [34], then there is no computation or transmission interference. He has also studied several variations of his rules and the corresponding classes of trace structures.

A different formalization of the foam-rubber wrapper postulate was developed by H. Schols in 1985, in terms of a certain composition operation on trace structures [28]. This formalization does not include the absence of

transmission interference, since there exist trace structures that satisfy the foam-rubber wrapper postulate of Schols, but fail Udding's Rule 1.

Delay-insensitivity of arbiters satisfying fairness conditions was studied by A. J. Martin [21] (1985), D. L. Black [1] (1986), and J. T. Udding [35, 36] (1985–86). The conclusions about the existence or non-existence of delay-insensitive arbiters depend on the definition of fairness and delay-insensitivity.

In 1987 J. C. Ebergen studied decompositions of asynchronous components (behaviors) specified by trace structures into delay-insensitive connections of basic components [13, 14]. A component $S.0$ can be *decomposed* into components $S.i$, $i = 1, \ldots, n$ if the following conditions (stated only informally here) are met:

1. The system $\{S.i \mid i = 0, \ldots, n\}$ is closed, in the sense that every output of every component is connected to an input of another component, and the same is true if "input" and "output" are interchanged.

2. No two outputs are connected together.

3. The connection behaves as specified by $S.0$. (The environment of the connection of the n components is assumed to be $S.0$.)

4. There is no computation interference.

Ebergen's work can be viewed as another formalization of the foam-rubber wrapper postulate. Informally, a decomposition is delay-insensitive, if it is still a decomposition (in the sense given above) after wire delays have been added. A component S is said to be delay-insensitive, if it operates correctly in the presence of wire delays. Ebergen derives properties of decompositions that permit the decomposition of a specification in a hierarchical way, and that enable us to decompose parts of a specification separately. He proves that, if the basic components are delay-insensitive, then the concept of decomposition coincides with the concept of delay-insensitive decomposition. Ebergen proves that his definition of delay-insensitive component is equivalent to Udding's definition. Thus, a component is delay-insensitive in Ebergen's sense if and only if it satisfies Udding's four rules. Ebergen also develops systematic methods for decomposing a specification into a network of basic components.

T.-A. Chu developed (1987) methods of designing speed-independent circuits from *signal transition graphs,* which are a form of interpreted Petri

nets [11]. A condition closely related to semi-modularity is called *persistence* by Chu. Signal transition methods have been widely used; see, for example, [19].

In 1988, D. L. Dill developed methods for the specification and automatic verification of asynchronous circuits [12]. He used the term "speed-independent" in an informal sense, but he did introduce a formal definition of delay-insensitivity. Dill uses a modified trace theory for behavior specifications. Dill's trace structure is a 4-tuple $T = \langle \mathbf{i}T, \mathbf{o}T, \mathbf{s}T, \mathbf{d}T \rangle$, where $\mathbf{i}T$ and $\mathbf{o}T$ are finite disjoint sets of *inputs* and *outputs*, $\mathbf{s}T$ is a set of *successes* (successful traces), and $\mathbf{d}T$ is a set of *divergences* (called *failures* by Dill). The sets $\mathbf{s}T$ and $\mathbf{d}T$ are (not necessarily disjoint) prefix-closed, regular sets of traces over the alphabet $\mathbf{a}T = \mathbf{i}T \cup \mathbf{o}T$. The set $\mathbf{p}T = \mathbf{s}T \cup \mathbf{d}T$ is the set of *possible* traces. This set is nonempty, since the empty trace is always possible. The traces in $(\mathbf{a}T)^* - \mathbf{p}T$ represents the set of *impossible* traces. Trace structures must be *receptive*, meaning that $(\mathbf{p}T)(\mathbf{i}T) \subseteq \mathbf{p}T$. Thus, if a trace w is possible, and x is a module input, then trace wx is possible because the module cannot prevent the environment from sending a signal.

We illustrate this model for a logic gate. If the gate is unstable, and the input changes, making the gate stable, the corresponding trace is a divergence (failure), because it corresponds to a hazard. However, an input may change successfully if the gate is unstable before and after the change.

Dill defines the concept "trace structure T conforms to trace structure S" as follows. Suppose that S forms part of a network N that operates correctly in the sense that the failure set of its trace structure is empty. Then T *conforms to* S *in* N if the failure set of the network in which S has been replaced by T is still empty. Then T *conforms* to S if it conforms to S in all correct networks. Next, Dill defines the operator DI on trace structures. Intuitively, DI adds delays to all the input and output terminals of a trace structure, hides the original terminals, and renames the new inputs and outputs to the original names. This corresponds to adding a foam-rubber wrapper to the module. A trace structure T is said to be delay-insensitive if and only if $DI(T)$ conforms to T.

In 1988 C-J. Seger gave a new proof of Unger's theorem on essential hazards [7, 30, 31]. This proof is discussed further in Section 6.

In 1989 E. Brunvand and R. F. Sproull [2] developed a module-based approach to the design of delay-insensitive circuits. They used a subset of the language OCCAM to specify behaviors, and designed delay-insensitive modules for each language construct (for example, a *while* loop). This permits automatic compilation of programs to networks of modules. See also [3].

In 1990 A. J. Martin essentially used semi-modularity, called *stability* by him, as a *definition* of delay-insensitivity [22]. (J. A. Brzozowski and H. Zhang have proved recently that this assumption can, indeed, be justifiable [9]. This is briefly discussed later.) In [22], Martin considers limitations of a restricted class of delay-insensitive circuits. The modules used by him include not only logic gates, but also some sequential modules such as C-elements. The networks Martin considers are autonomous.

In 1992 J. A. Brzozowski and J. C. Ebergen showed that very few behaviors are implementable delay-insensitively, if the circuits are operated in the so-called input/output mode [5]. These results were somewhat generalized in [7], and are discussed in Section 7.

N. Shintel and M. Yoeli specified (1992) behaviors by Petri nets [33]. Given a network N of modules they defined the *delay-extension* of N as the network $\hat{N}$ obtained from N by adding delays to the input and output wires of each module. Roughly speaking, they define a network N to be an implementation of a behavior $\mathcal{B}$ if the trace set of N is the same as that of $\mathcal{B}$, and also N does not deadlock, in the sense that it fails to produce an output when one is expected according to $\mathcal{B}$. A network is considered to be a delay-insensitive implementation of a behavior $\mathcal{B}$ if the delay-extension of the network is an implementation of $\mathcal{B}$.

H. Schols studied (1992) delay-insensitive communication [29]. He proposed another definition of delay-insensitivity in a somewhat different formalism, and argued that it his definition is equivalent to those of Udding and Ebergen.

In 1993 K. van Berkel studied delay-insensitive *handshake processes* [39]. The alphabets of handshake processes have more structure than those of trace structures, and the trace set is not prefix-closed. However, for every handshake process, one can define a corresponding trace structure. A handshake process is then considered delay-insensitive if its corresponding trace structure is delay-insensitive. Van Berkel proves that his handshake processes are delay-insensitive.

L. Lavagno and A. Sangiovanni-Vincentelli developed (1993) a methodology for asynchronous circuit design [19]. They specify behaviors by signal transition graphs, and assume that wire delays are bounded. The book [19] contains an extensive survey of previous work on asynchronous circuits. Signal transition graphs are classified "speed-independent with inertial delays," "speed-independent with pure delay," or delay-insensitive in Udding's sense. Semi-modularity and related properties are also considered.

In 1994 T. Verhoeff extended Uddings rules to include progress as a correctness concern, in addition to absence of interference [42]. In Verhoeff's terminology, a *process* is identified with a trace structure. An *(extended) process* is a quintuple $T = \langle \mathbf{i}T, \mathbf{o}T, \nabla T, \Box T, \triangle T \rangle$, where $\mathbf{i}T$ and $\mathbf{o}T$ are disjoint input and output alphabets as before, and ∇T, $\Box T$, and $\triangle T$ are pairwise disjoint trace sets with the properties given below. Traces in ∇T are *transient* traces; the upside down triangle suggests that the process state corresponding to a transient trace is unstable (will eventually topple) and will be followed by an output. For each transient trace w, there exists an output symbol $z \in \mathbf{o}T$ such that wz is in $\mathbf{t}T$, where $\mathbf{t}T = \nabla T \cup \Box T \cup \triangle T$ is the set of traces of T. Traces in $\triangle T$ are *input-demanding* traces. For each input-demanding trace there exists an input symbol $x \in \mathbf{o}T$ such that wx is in $\mathbf{t}T$. States corresponding to input-demanding traces can persist indefinitely, if the environment fails to supply an input (the base of the triangle is stable). However, the environment is expected to produce an input for an input-demanding trace. Traces in $\Box T$ are *indifferent*; after such a state, neither the process nor the environment are obliged to make progress.

Verhoeff generalized Udding's rules to extended processes. Rules 1 and 4 remain unchanged; Rules 2, and 3 require modifications. Rule 2 becomes: For any $u, v \in \mathbf{t}T$, and a and b of the same type,

- $uabv \in \mathbf{t}T$ if and only if $ubav \in \mathbf{t}T$,
- $uabv \in \nabla T$ if and only if $ubav \in \nabla T$,
- $uabv \in \triangle T$ if and only if $ubav \in \triangle T$.

Rule 3 requires a similar modification.

Verhoeff has also shown [41, 42] that Udding's Rules 2, 3, and 4 are equivalent to a single rule, which he calls the "neighbor-swap rule." Space limitations prevent us from providing further details; we refer the reader to [41, 42], where several other interesting characterizations of delay-insensitivity are given.

Another algebraic study of delay-insensitive processes has been carried out by P. G. Lucassen in 1994 [20]. He views delay-insensitive processes as special kinds of *receptive processes* developed in 1992 by M. B. Josephs to model the behaviors of speed-independent circuits [17]. Here, a process is an input alphabet, an output alphabet, and an algebraic expression that describes the communications on the input and output ports of the process.

M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky published a book (1994) on an asynchronous design approach based on *change diagrams*, which are graphs similar to signal transition graphs [18]. For semi-modular circuits, the two models have similar modeling power.

In 1995 a survey of several asynchronous design methodologies was written by S. Hauck [16]; a revised version by J. A. Brzozowski, S. Hauck, and C-J. Seger appeared as Chapter 15 of [7].

In 1996 M. E. Bush and M. B. Josephs studied the limitations of speed-independent circuits [10] in the formalism of receptive processes [17]. Like many others, Bush and Josephs use trace structures $T = \langle \mathbf{i}T, \mathbf{o}T, \mathbf{t}T \rangle$; however, they introduce more structure. The set $\mathbf{f}T \subseteq \mathbf{t}T$ consists of *failures* that may take the circuit to an undesirable state or to a quiescent state from which no output is issued. The set $\mathbf{d}T \subseteq \mathbf{f}T$ consists only of traces leading to undesirable states, and is called the set of *divergences* of T. Bush and Josephs also add Dill's receptivity condition, as described above. The authors formulate five properties that should be satisfied by speed-independent circuits. These properties are weaker than those of Udding [36]. In all the items below, a is an input, b is either an input or an output, c is an output, and u and v are traces. The rules are

1. An enabled event should remain enabled after the occurrence of an input. Formally, if $ub \in \mathbf{t}T$ then $uab \in \mathbf{t}T$.

2. If $uba \notin \mathbf{d}T$, then $ubav \in \mathbf{f}T$ implies $uabv \in \mathbf{f}T$.

3. If an input is unsafe after an output, then it is also unsafe before that output. Formally, if $uca \in \mathbf{d}T$, then $ua \in \mathbf{d}T$.

4. If $uv \in \mathbf{f}T$ then $uaav \in \mathbf{f}T$.

5. If $uaa \notin \mathbf{d}T$, then $uv \in \mathbf{f}T$ if and only if $uaav \in \mathbf{f}T$.

In 1997 J. A. Brzozowski and H. Zhang proved that in a certain network model, which includes wire delays, delay-insensitivity corresponds precisely to semi-modularity [9]. A very general model of a sequential module, namely a Moore type sequential machine [27], is used. Such modules include delays, forks, logic gates, latches, counters, C-elements, and arbiters. A network N of such modules is delay-insensitive if the behavior of any network $\hat{N}$ derived from N by the addition of delays is bisimilar [25] to the behavior of N. In effect, N and $\hat{N}$ can always simulate each other, if one looks only on the state variables of N. Note that this condition is stronger than conditions based

on standard trace theory; for example, it permits the handling of deadlock. It is proved that a network is delay-insensitive if and only if its behavior is semi-modular, where the concept of semi-modularity is generalized to permit nondeterminism.

3 Gate Networks

We now introduce our model of gate networks. This model is based on that of [9], but is much simpler, since we only consider logic gates.

A *combinational module* is an ordered pair $M^i = \langle C^i, G^i \rangle$, where

- $C^i = \langle \mathcal{X}^i, s^i, \mathcal{Z}^i, S^i \rangle$ is a *gate*, where $\mathcal{X}^i$, $\{s^i\}$, and $\mathcal{Z}^i$ are pairwise disjoint sets, and
  - $\mathcal{X}^i = \{x^i_1, \ldots, x^i_{m^i}\}$, $m^i \geq 0$, is the set of *module input variables*; also, $x^i = (x^i_1, \ldots, x^i_{m^i})$ is the m^i-tuple of module input variables;
  - s^i is the *module internal-state variable*;
  - $\mathcal{Z}^i = \{z^i_1, \ldots, z^i_{p^i}\}$, $p^i \geq 0$, is the set of *module output variables*; also, $z^i = (z^i_1, \ldots, z^i_{p^i})$ is the p^i-tuple of module output variables;
  - S^i is the *module excitation function*, $S^i : \{0,1\}^{m^i} \rightarrow \{0,1\}$;
- $G^i = \langle \mathcal{V}^i, \mathcal{E}^i \rangle$ is a directed graph, the *module graph*, where
  - $\mathcal{V}^i = \mathcal{X}^i \cup \{s^i\} \cup \mathcal{Z}^i$ is the set of *module vertices*, where
    - $\mathcal{X}^i$ is the set of *module input vertices*, each with indegree 0 and outdegree 1;
    - s^i is the *module internal-state vertex* with indegree m^i and outdegree p^i;
    - $\mathcal{Z}^i$ is the set of *module output vertices*, each with indegree 1 and outdegree 0;
  - $\mathcal{E}^i = (\mathcal{X}^i \times \{s^i\}) \cup (\{s^i\} \times \mathcal{Z}^i)$ is the set of *module edges*. □

Combinational modules include delays (or wires with delays), forks, and logic gates performing arbitrary Boolean functions and having multiple outputs. All the output variables always have the same value as the state variable; thus the multiple outputs are used simply to provide fan-out. Note that

the forks in this model are *isochronic*, meaning that all the output branches change at exactly the same time, simultaneously with internal state change.

We illustrate this definition with some examples. For simplicity, the superscript i is omitted.

- *Delay:* $M = \langle C, G \rangle$, $C = \langle \{x_1\}, s, \{z_1\}, S \rangle$, and $S(x_1) = x_1$.
- *Fork:* $M = \langle C, G \rangle$, $C = \langle \{x_1\}, s, \{z_1, z_2\}, S \rangle$, and $S(x_1) = x_1$.
- *Majority gate:* $M = \langle C, G \rangle$, $C = \langle \{x_1, x_2, x_3\}, s, \{z_1\}, S \rangle$, S is given by the Boolean expression $S(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_3x_1$.

A *network* is an ordered pair $N = \langle \mathcal{M}, G \rangle$, where

- $\mathcal{M} = \{M^1, \ldots, M^n\}$, $n \geq 0$, is a set of modules.
- $G = \langle \mathcal{V}, \mathcal{E} \rangle$ is a directed graph, where
  - $\mathcal{V} = \mathcal{X} \cup \mathcal{I} \cup \mathcal{Z}$ is the set of *network vertices*, where $\mathcal{X}$, $\mathcal{I}$, and $\mathcal{Z}$ are pairwise disjoint, and
    - $\mathcal{X} = \{x_1, \ldots, x_m\}$ is the set of *network input vertices*, each with indegree 0 and outdegree 1;
    - $\mathcal{I} = \bigcup_{i=1}^{n} \mathcal{V}^i$ is the set of *network internal vertices*;
    - $\mathcal{Z} = \{z_1, \ldots, z_p\}$ is the set of *network output vertices*, each with indegree 1 and outdegree 0;
  - $\mathcal{E}$ is the set of edges, $\mathcal{E} = \bigcup_{i=1}^{n} \mathcal{E}^i \cup \mathcal{K}$, where

$$\mathcal{K} \subseteq \left(\mathcal{X} \times \bigcup_{i=1}^{n} \mathcal{X}^i \right) \cup \left(\bigcup_{i=1}^{n} \mathcal{Z}^i \times \bigcup_{i=1}^{n} \mathcal{X}^i \right) \cup \left(\bigcup_{i=1}^{n} \mathcal{Z}^i \times \mathcal{Z} \right),$$

    is the set of *connections* of N. Furthermore, each vertex in $\mathcal{X}$ and in each $\mathcal{Z}^i$ has exactly one outgoing edge, and each vertex in each $\mathcal{X}^i$ and in $\mathcal{Z}$ has exactly one incoming edge.

The set $\mathcal{S} = \{s^1, \ldots, s^n\}$ is called the set of *network state variables*. The set $\delta = (\delta_1, \ldots, \delta_n)$, is the *network excitation vector*, where δ_i is the excitation function S^i of module M^i with arguments changed as follows. If (z_h^j, x_k^i) is a connection, then the kth argument of δ_i is s^j. If (x_j, x_k^i) is a connection, then the kth argument of δ_i is x_j.

To take into account wire delays, we can insert a delay module in any connection of a network as follows. Suppose $e = (v_i, v_j)$ is a connection

edge in which we wish to insert a delay module $M^d = \langle C^d, G^d \rangle$, where $C^d = \langle \{x_1^d\}, s^d, \{z_1^d\}, S^d \rangle$. We add M^d to the set of modules of N. We add vertices x_1^d, s^d, and z_1^d to the set of vertices of N. We remove edge e, and add edges (v_i, x_1^d) and (z_1^d, v_j). Finally, we add the excitation δ^d to the vector of excitations.

4 Autonomous Behaviors of Gate Networks

In this section we analyze the behavior of a gate network in the binary general multiple-winner (GMW) model, assuming that the network input tuple has some fixed binary value. This material is based on [7, 8, 24].

An assignment of a binary m-tuple a to the m-tuple x of network input variables is an *input state*. An assignment of a binary n-tuple $b \in \{0,1\}^n$ to the n-tuple s of state variables is an *internal state*. The pair (a, b) is a *total state*. The *excitation* of N in total state (a, b) is the n-tuple $B = B_1, \ldots, B_n = \delta_1(a, b), \ldots, \delta_n(a, b)$. A state b is *stable* if $B_i = b_i$ for all i; otherwise, b is *unstable*. The set

$$\mathcal{U}(b) = \{s^i \in \mathcal{S} \mid B_i \neq b_i\}$$

is the set of unstable variables in state b.

Assume that the input m-tuple is fixed at $x = a$. The *GMW behavior* of a network N started in state q and having fixed input a is an initialized directed graph $\mathcal{B}_a(q) = \langle q, \mathcal{Q}, R_a \rangle$, where $q \in \{0,1\}^n$ is the *initial state*, $\mathcal{Q} = \mathit{reach}(\mathcal{B}_a(q))$ is the set of states reachable from q by using the GMW relation R_a, where R_a is defined by bR_ac if b is stable and $b = c$, or c is obtained from b by complementing all the variables in some nonempty set of unstable variables. The state variables that are unstable in a given state are in a *race* to change. Allowing several variables to change corresponds to multiple winners of the race. Since this model makes no assumptions about the relative sizes of delays, it is the "general multiple-winner" model.

Since the behavior graph is finite, every path from q must eventually reach a cycle. A cycle in $\mathcal{B}$ is *transient* if there exists a state variable s_i that has the same value in all the states of the cycle and is unstable in each state of the cycle. A behavior cannot remain in a transient cycle longer than the delay of the variable s_i. Let the set of *cyclic states* reachable from q in the behavior graph of $\mathcal{B}_a(q)$ be

$$\mathit{cycl}(\mathcal{B}_a(q)) = \{b \in \{0,1\}^m \mid qR_a^*b \text{ and } bR_a^+b\},$$

where R_a^+ is the transitive closure of R_a, and R_a^* is the reflexive and transitive closure of R_a. Next, define the set of nontransient cyclic states to be

$$cycl_nontrans(\mathcal{B}_a(q)) = \{b \mid b \text{ appears in a nontransient cycle}\}.$$

The *outcome* of the transition from q under input a is the set of all the states that are reachable from a state in a nontransient cycle. Mathematically, we have

$$out\,(\mathcal{B}_a(q)) = \{b \mid qR_a^*c \text{ and } cR_a^*b, \text{ where } c \in cycl_nontrans(\mathcal{B}_a(q))\}.$$

Like Muller's terminal class [24], the outcome consists of all the states that represent the "final destination" of the behavior. In the presence of arbitrary, but bounded, gate and wire delays, for any sufficiently long time after the network leaves state q, it must be in one of the states in the outcome [7].

5 Ternary Simulation

This section is based on [6, 7, 15, 32]. Ternary simulation, originally introduced by E. B. Eichelberger, is an analysis method based on three logic values, 0, 1, and Φ. The three values are partially ordered by the *uncertainty partial order* $\sqsubseteq$, defined as

$$0 \sqsubseteq 0,\ 1 \sqsubseteq 1,\ \Phi \sqsubseteq \Phi,\ 0 \sqsubseteq \Phi, \text{ and } 1 \sqsubseteq \Phi,$$

and no other pairs are related by $\sqsubseteq$. The values 0 and 1 are thought of as *certain*, whereas Φ is the *uncertain* value. The partial order $\sqsubseteq$ is extended to $\{0, \Phi, 1\}^n$ in the natural way. Also, the concept of least upper bound *lub* is defined for this partial order as usual.

Instead of using the Boolean module excitation functions δ_i, we use their ternary extensions $\boldsymbol{\delta}_i$. For a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$, its *ternary extension* $\mathbf{f}\colon \{0, \Phi, 1\}^n \to \{0, \Phi, 1\}$ is

$$\mathbf{f}(\mathbf{a}) = lub\,\{f(b) \mid b \in \{0,1\}^n \text{ and } b \sqsubseteq \mathbf{a}\},$$

for all $\mathbf{a} \in \{0, \Phi, 1\}^n$. Note that any Boolean function f agrees with its ternary extension $\mathbf{f}$ when the argument $\mathbf{a}$ is binary.

A network using the binary domain will be denoted by N, whereas its ternary counterpart will be $\mathbf{N}$.

Ternary simulation consists of two algorithms, A and B. The network is assumed to be started in total state (a, q). In the first algorithm, the variables of unstable modules are assigned the value Φ. This introduces some uncertain values into the network, and this uncertainty may then spread to other parts of the network. Algorithm A is formally defined as follows:

Algorithm A
$h := 0$;
$\mathbf{s}^0 := q$;
repeat
  $h := h + 1$;
  $\mathbf{s}^h := lub\,\{\mathbf{s}^{h-1}, \boldsymbol{\delta}(a, \mathbf{s}^{h-1})\}$;
until $\mathbf{s}^h = \mathbf{s}^{h-1}$;

Proposition 1 *Algorithm* A *generates a finite sequence* $\mathbf{s}^0, \ldots, \mathbf{s}^A$ *of states, where* $A \leq n$. *Furthermore, this sequence is monotonically increasing, i.e.,*

$$\mathbf{s}^h \sqsubset \mathbf{s}^{h+1},\ for\ 0 \leq h < A.$$

The final outcome of Algorithm A is the ternary state $\mathbf{s}^A$.

In Algorithm B, some of the uncertainty introduced by Algorithm A may be removed; we start the network in the state generated by Algorithm A.

Algorithm B
$h := 0$;
$\mathbf{t}^0 := \mathbf{s}^A$;
repeat
  $h := h + 1$;
  $\mathbf{t}^h := \boldsymbol{\delta}(a, \mathbf{t}^{h-1})$;
until $\mathbf{t}^h = \mathbf{t}^{h-1}$;

Proposition 2 *Algorithm* B *generates a finite sequence* $\mathbf{t}^0, \ldots, \mathbf{t}^B$ *of states, where* $B \leq n$. *Furthermore, this sequence is monotonically decreasing, i.e.,*

$$\mathbf{t}^h \sqsupset \mathbf{t}^{h+1},\ for\ 0 \leq h < B.$$

The following two theorems are the fundamental results relating GMW analysis and ternary simulation. A network is said to be *delay-complete* if a delay is included for each network input vertex, and for each connection, as described above. In other words, we use the input-, gate-, and wire-delay model. Input delays are used for mathematical convenience.

Theorem 1 *Let $N = \langle \mathcal{M}, G \rangle$ be a complete binary network, and let $\mathbf{N} = \langle \mathbf{M}, G \rangle$ be its ternary counterpart. If N is started in total state (a, q), then the result $\mathbf{s}^A$ of Algorithm* A *for $\mathbf{N}$ is equal to the lub of the set $\mathcal{Q}$ of all the states reachable from the initial state in the GMW behavior of N, i.e.,*

$$\mathbf{s}^A = lub\, \mathcal{Q}.$$

Theorem 2 *Let $N = \langle \mathcal{M}, G \rangle$ be a complete binary network, and let $\mathbf{N} = \langle \mathbf{M}, G \rangle$ be its ternary counterpart. If N is started in total state (a, q), then the result $\mathbf{t}^B$ of Algorithm* B *is equal to the lub of the outcome of the GMW analysis, i.e.,*

$$\mathbf{t}^B = lub\, out\,(\mathcal{B}_a(q)).$$

6 Delay-Insensitivity in Fundamental Mode

For many applications, the desired behavior of an asynchronous network consists of a series of transitions among stable states. To be more precise, we assume that the network starts in some initial state (a^0, b^0). The environment then changes the input to a^1, say, and waits for the circuit to reach a stable state (a^1, b^1). The environment may then change the input to a^2, etc. In general, a network is said to be operated in *fundamental mode* if its inputs are permitted to change only if the entire network is stable. Notice that two assumptions are requireed for this definition. First it is assumed that the network does, indeed, reach a unique stable state after every input change. Thus, a behavior in which two different stable states could be reached, depending on the relative sizes of the network delays, is considered erroneous. Also, oscillations are considered undesirable. Second, one has to have an estimate of the time required for a network to reach a stable state after any input change. The environment of the network then waits for that time to elapse, before sending new inputs.

Formally, the analysis of a network operated in fundamental mode consists of a series of computations of the outcome using GMW analysis. A transition of a network N, from a stable state $(\hat{a}, b)$ under new input vector a, is said to be *delay-insensitive in fundamental mode* if and only if $out\,(\mathcal{B}_a(b))$ contains a single state, where $out\,(\mathcal{B}_a(b))$ is the outcome of the GMW analysis in the input-, gate-, and wire-delay model. Note that this definition involves only a rather crude correctness criterion. We are interested only in the final state reached, and not in any intermediate states. But, even with this rough criterion, delay-insensitive behaviors are very limited.

In particular, an old result due to Unger [37, 38] states that a behavior is not delay-insensitively implementable by a gate network if it contains a so-called essential hazard. Suppose a behavior is in a stable state $(\hat{a}, b)$ and the input changes first to a, then back to $\hat{a}$, and again to a. The behavior has an essential hazard if the state reached after the first input change is different from that reached after the third input change. A new proof of this result in a formal model has been given by Seger [7, 30, 31]. This proof is based on Theorems 1 and 2.

Theorem 3 *Let N be any network and let $(\hat{a}, \hat{b})$ be a stable state of N. If $(\hat{a}, \hat{b})$ to (a, b), and (a, b) to $(\hat{a}, \tilde{b})$ are delay-insensitive transitions of N, then so is the transition from $(\hat{a}, \tilde{b})$ under input a, and the result of this transition is again the state (a, b).*

7 Delay-Insensitivity in Input/Output Mode

Informally, a network is operated in *input/output mode* [4, 5, 7] if the environment is allowed to change the network's input only after receiving an appropriate output signal from the network. Unfortunately, there is no general formal definition of input/output mode, and this area needs further work. For our purposes, we will only require some rather simple examples. In fact, it has been shown [5, 7] that the simple behavior given below has no delay-insensitive input/output mode implementation. We use the concept of *whole state* to represent asynchronous behaviors. A whole state is a total state together with the output; since the output is determined by total state, this is redundant, but we use it for convenience. We denote a whole state (a, b, c) by $a{\cdot}b{\cdot}c$.

Consider a behavior $\mathcal{B}_0$ with one binary input x, one binary output z, and two internal states 0 and 1. The output is always equal to the internal state.

$$0{\cdot}0{\cdot}0 \xrightarrow{\{x\}} 1{\cdot}0{\cdot}0 \xrightarrow{\{z\}} 1{\cdot}1{\cdot}1 \xrightarrow{\{x\}} 0{\cdot}1{\cdot}1.$$

The behavior starts in stable whole state $0{\cdot}0{\cdot}0$. When the input changes to 1, the behavior should respond by changing its output to 1. After that, the environment can change the input back to 0, and the behavior should never again change its output. Note that this behavior is delay-insensitively implemented in fundamental mode by an OR gate with output z and inputs x and z. The situation changes, however, if input/output mode is used.

The following result was first proved in [5], and presented in a slightly more general framework in [7]:

Theorem 4 *Behavior $\mathcal{B}_0$ has no delay-insensitive implementation by a gate circuit operated in input/output mode.*

In spite of the simplicity of the behavior, it is not easy to prove this theorem. The problem, of course, is that we must show that no matter how we construct a gate network, it can never behave like $\mathcal{B}_0$. Here again the ternary simulation theorems come to the rescue.

First, we need to establish some general properties of the proposed solution. Suppose that a network N implementing $\mathcal{B}_0$ exists. Then this network must have one binary input x, one binary output z, and some number of internal states. There must be at least one internal state, say b, that represents the initial state, and the output in that state must be 0. Thus, we can assume that our network N starts in whole state $0{\cdot}b{\cdot}0$. We are not allowed to assume that this state is stable. In fact, N could be in an oscillation; as long as the output is 0, the behavior is correct. To cover this situation, we introduce a typical state $0{\cdot}c{\cdot}0$ to which a transition from $0{\cdot}b{\cdot}0$ is allowed. Now, in any state like $0{\cdot}c{\cdot}0$, the environment is permitted to change the input to 1; this must result in some state $1{\cdot}d{\cdot}0$. Note that, since we know nothing about the internal state of N, we allow it to change at the same time as the input changes. We do know, however, that this last state $1{\cdot}d{\cdot}0$ cannot be stable, because an output change must be produced. Some state (or states) of the form $1{\cdot}e{\cdot}0$ might be entered, but a state of the form $1{\cdot}f{\cdot}1$ must eventually be reached. In summary, our network N must have the following type of behavior corresponding to the first input change:

$$0{\cdot}b{\cdot}0 \longrightarrow 0{\cdot}c{\cdot}0 \xrightarrow{\{x\}} 1{\cdot}d{\cdot}0 \longrightarrow 1{\cdot}e{\cdot}0 \xrightarrow{\{z\}} 1{\cdot}f{\cdot}1.$$

Once the output changes, the environment is allowed to change x back to 0, resulting in some state $0{\cdot}h{\cdot}1$. After that, the internal state of N may change again, but never its output. Thus, we may have another typical state $0{\cdot}i{\cdot}1$. The possible state sequence is:

$$0{\cdot}b{\cdot}0 \longrightarrow 0{\cdot}c{\cdot}0 \xrightarrow{\{x\}} 1{\cdot}d{\cdot}0 \longrightarrow 1{\cdot}e{\cdot}0 \xrightarrow{\{z\}} 1{\cdot}f{\cdot}1 \longrightarrow 1{\cdot}g{\cdot}1 \xrightarrow{\{x\}} 0{\cdot}h{\cdot}1 \longrightarrow 0{\cdot}i{\cdot}1.$$

Note that many other internal states may occur; we only show what might be typical.

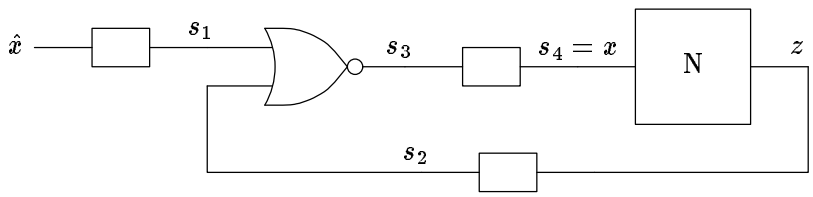

Figure 1: Network $\hat{N}$.

Assuming that N exists, we construct a network $\hat{N}$ using N, as shown in Figure 1. First, we find the GMW behavior of $\hat{N}$. Let the total state of $\hat{N}$ be $\hat{x}{\cdot}s_1s_2s_3s_4y{\cdot}z$, where $s_4 = x$ is the input to N, y is its internal state, and z is its output. Note that the output of N must be derived from some delay, since we are using the input-, gate-, and wire-delay model for N; thus, it is part of the state. (Formally, we should add another delay between $\hat{x}$ and the s_1 input of the NOR gate, but this would not change the argument.) Network $\hat{N}$ will be operated with a constant input. It is started in state $0{\cdot}1000b{\cdot}0$, that is, the input delay is unstable, and N is in its initial state $0{\cdot}b{\cdot}0$. It is clear that $\hat{N}$ must reach a state of the form $0{\cdot}0011d{\cdot}0$. According to the behavior of N, we must eventually reach a state of the form $1{\cdot}f{\cdot}1$; hence, $\hat{N}$ reaches $0{\cdot}0011f{\cdot}1$. Next, delay s_2 will change to 1, changing s_3 and s_4 to 0. According to the behavior of N, there will not be any observable changes after that. Hence, the outcome of this transition of $\hat{N}$ must be a state (or states) of the form $0{\cdot}0100i{\cdot}1$. The important point to notice is that GMW analysis predicts 1 as the final value of z.

Suppose now that we analyze $\hat{N}$ using ternary simulation. The state sequence for Algorithm A is

$$0{\cdot}1000b{\cdot}0 \longrightarrow 0{\cdot}\Phi000y{\cdot}0 \longrightarrow 0{\cdot}\Phi0\Phi0y{\cdot}0 \longrightarrow 0{\cdot}\Phi0\Phi\Phi y{\cdot}0,$$

where y is the unknown (and irrelevant) state of N. Since we know that both state $0{\cdot}1000b{\cdot}0$ and state $0{\cdot}0011f{\cdot}1$ are reachable in the GMW analysis of $\hat{N}$, we know by Theorem 1 that the final value of z in Algorithm A is Φ. Thus the result of Algorithm A must be $0{\cdot}\Phi\Phi\Phi\Phi y{\cdot}\Phi$.

If we next apply Algorithm B starting with state $0{\cdot}\Phi\Phi\Phi\Phi y{\cdot}\Phi$, we obtain the result $0{\cdot}0\Phi\Phi\Phi y{\cdot}\Phi$. By Theorem 2 we know that there must be a state in the GMW outcome with $z = 0$. This contradicts the assumption that z remains at 1, and proves that network N cannot exist.

It is an immediate consequence of Theorem 4 that the behaviors of many common components like JOIN, TOGGLE, latch cannot be implemented delay-insensitively in the input/output mode [5, 7]. Moreover, Theorem 4 can be generalized as follows. A behavior is *simple deterministic* [7] if it is a fundamental-mode behavior, it is deterministic, every transition from an unstable state goes directly to a stable state, and at most one input or output variable changes in any transition. Such a behavior is *nontrivial* if there exits at least one input state a and at least two stable whole states $a{\cdot}b{\cdot}c$ and $a{\cdot}b'{\cdot}c'$ such that $c \neq c'$ and one of the states is reachable from the other. Using both Theorem 3 and 4, it is possible to prove the following [5]:

Theorem 5 *No nontrivial simple deterministic behavior with a binary input has a delay-insensitive implementation by a gate circuit operated in input/output mode.*

Using Theorem 2, it is easy to prove the following result concerning arbiters [7, 32]:

Theorem 6 *No arbiter has a delay-insensitive implementation by a gate circuit operated in input/output mode.*

8 Conclusions

We have shown that there are many ways of defining delay-insensitivity. One of the minimal requirements we can impose as the correctess of a network is that it makes a transition to the correct final state. Even with this simple concern, very few behaviors have delay-insensitive implementations by gate circuits. The proofs of several such results are possible if we use the result linking GMW analysis to ternary simulation.

Acknowledgment The author thanks David Dill, Jo Ebergen, Luciano Lavagno, Jan Tijmen Udding, Tom Verhoeff, and Hao Zhang for their comments on this paper, particularly on Section 2.

References

[1] D. L. Black, "On the Existence of Delay-Insensitive Fair Arbiters: Trace Theory and its Limitations," *Distributed Computing,* vol. 1, no. 4, pp. 205–225, 1986.

[2] E. Brunvand and R. F. Sproull, "Translating Concurrent Programs into Delay-Insensitive Circuits," *Proc. of International Conference on Computer-Aided Design,* IEEE Press, pp. 262–265, 1989.

[3] E. Brunvand, *Translating Concurrent Communicating Programs into Asynchronous Circuits,* PhD Thesis, Computer Science Department, Carnegie Mellon University, 1991. Also, Tech. Rept. CMU–CS–91–198.

[4] J. A. Brzozowski and J. C. Ebergen, "Recent Developments in the Design of Asynchronous Circuits," *Proc. of Fundamentals of Computation Theory,* J. Csirik, J. Demetrovics, and F. Gécseg, eds., Lecture Notes in Computer Science, No. 380, Springer-Verlag, Berlin, pp. 78–94, 1989.

[5] J. A. Brzozowski and J. C. Ebergen, "On the Delay-Sensitivity of Gate Networks," *IEEE Trans. on Computers,* vol. 41, no. 11, pp. 1349–1360, November 1992.

[6] J. A. Brzozowski and C-J. Seger, "A Characterization of Ternary Simulation of Gate Networks," *IEEE Trans. on Computers,* vol. C–36, no. 11, pp. 1318–1327, November 1987.

[7] J. A. Brzozowski and C-J. Seger, *Asynchronous Circuits,* Springer-Verlag, New York, NY, 1995.

[8] J. A. Brzozowski and M. Yoeli, "On a Ternary Model of Gate Networks," *IEEE Trans. on Computers,* vol. C–28, no. 3, pp. 178–183, March 1979.

[9] J. A. Brzozowski and H. Zhang, "Delay-Insensitivity and Semi-Modularity," Research Report CS–97–11, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, March 1997.

[10] M. E. Bush and M. B. Josephs, "Some Limitations to Speed-Independence in Asynchronous Circuits," *Proc. Second International Symposium on Advanced Research in Asynchronous Circuits and Systems,* March 18–21, 1996, Aizu-Wakamatsu, Japan, IEEE Computer Society Press, Los Alamitos, CA, pp. 104–111, 1996.

[11] T.-A. Chu, *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications,* PhD Thesis, Dept. of Electrical Engineering and Computer Science, Massachusets Institute of Technology, June 1987.

[12] D. L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits,* PhD Thesis, Computer Science Department, Carnegie Mellon University, February 1988. Also, The MIT Press, Cambridge, Massachusets, 1989.

[13] J. C. Ebergen, *Translating Programs into Delay-Insensitive Circuits,* PhD Thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, October 1987. Also, CWI Tract 56, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, 1989.

[14] J. C. Ebergen, "A Formal Approach to Designing Delay-Insensitive Circuits," *Distributed Computing,* vol. 5, no. 3, pp. 107–119, 1991.

[15] E. B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," *IBM J. Research and Development,* vol. 9, pp. 90–99, March 1965.

[16] S. Hauck, "Asynchronous Design Methodologies: An Overview," *Proc. of the IEEE,* vol. 83, no. 1, 1995.

[17] M. B. Josephs, "Receptive Process Theory," *Acta Informatica,* vol. 29, pp. 17–31, March 1992.

[18] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky, *Concurrent Hardware,* John Wiley & Sons, Chichester, England, 1994.

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

[20] P. G. Lucassen, *A Denotational Model and Composition Theorems for a Calculus of Delay-Insensitive Specifications,* PhD Thesis, Rijksuniversiteit Groningen, Groningen, The Netherlands, May 1994.

[21] A. J. Martin, "A Delay-Insensitive Fair Arbiter," Technical Report 5193:TR:85, Computer Science Department, California Institute of Technology, Pasadena, CA, 1985.

[22] A. J. Martin, "The Limitations to Delay-Insensitivity in Asynchronous Circuits," *Proc. of the Sixth MIT Conference on Advanced Research in VLSI,* W. J. Dally, ed., MIT Press, 1990.

[23] C. Mead and L. Conway, *Introduction to VLSI Systems,* Addison-Wesley, Reading, MA, 1980.

[24] R. E. Miller, *Switching Theory, Volume 2: Sequential Circuits and Machines,* John Wiley & Sons, Inc., New York, 1965.

[25] R. Milner, *Communication and Concurrency,* Prentice-Hall, 1989.

[26] C. E. Molnar, T. P. Fang, and F. U. Rosenberger, "Synthesis of Delay-Insensitive Modules," *Proc. of the 1985 Chapel Hill Conference on VLSI,* H. Fuchs, ed., Computer Science Press, Rockville, Maryland, pp. 67–86, 1985.

[27] E. F. Moore, "Gedanken Experiments on Sequential Machines," in C. E. Shannon and J. McCarthy, eds., *Automata Studies, Annals of Mathematics Study 34,* Princeton University Press, Princeton NJ, pp. 129–153, 1956.

[28] H. Schols, *A Formalisation of the Foam Rubber Wrapper Principle,* Master's Thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, February 1985.

[29] H. Schols, *Delay-Insensitive Communication,* PhD Thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, February 1985.

[30] C-J. H. Seger, *Models and Algorithms for Race Analysis in Asynchronous Circuits,* PhD Thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 186 pp., May 1988.

[31] C-J. H. Seger, "On the Existence of Speed-Independent Circuits," *Theoretical Computer Science,* vol. 86, no. 2, pp. 343–364, 1991.

[32] C-J. Seger and J. A. Brzozowski, "Generalized Ternary Simulation of Sequential Circuits," *Theoretical Informatics and Applications* vol. 28, no. 3/4, pp. 159–186, 1994.

[33] N. Shintel and M. Yoeli, "Synthesis of Modular Networks from Petri-Net Specifications," Technical Report #743, Computer Science Department, Technion, Haifa, Israel, June 1992.

[34] J. T. Udding, *Classification and Composition of Delay-Insensitive Circuits,* PhD Thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, September 1984.

[35] J. T. Udding, "On the Non-Existence of Delay-Insensitive Fair Arbiters," Technical Memorandum 306, Computer Systems Laboratory, Washington University, St. Louis, MO, July 1885.

[36] J. T. Udding, "A Formal Model for Defining and Classifying Delay-Insensitive Circuits and Systems," *Distributed Computing,* vol. 1, no. 4, pp. 197–204, 1986.

[37] S. H. Unger, "Hazards and Delays in Asynchronous Sequential Switching Circuits," *IRE Trans. on Circuit Theory,* vol. CT–6, pp. 12–25, March 1959.

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

[39] K. van Berkel, *Handshake Circuits,* Cambridge University Press, Cambridge, England, 1993.

[40] J. L. A. van de Snepscheut, *Trace Theory and VLSI Design,* PhD Thesis, Department of Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, May 1983. Also, *Lecture Notes in Computer Science,* vol. 200, Springer-Verlag, Berlin, 1985.

[41] T. Verhoeff, "Characterizations of Delay-Insensitive Communication Protocols," Computing Science Notes No. 89/06, Department of Mathematics and Computing Science, Eindhoven University of Technology, May 1989.

[42] T. Verhoeff, *A Theory of Delay-Insensitive Systems,* PhD Thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, May 1994.