Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 28, n° 3-4, 1994, p. 159 à 186)

GENERALIZED TERNARY SIMULATION OF SEQUENTIAL CIRCUITS (*)

by C.-J. SEGER ([1]) and J. A. BRZOZOWSKI ([2])

Abstract. – *Asynchronous gate circuits have been traditionally analyzed using binary models, which are conceptually simple and natural, but are exponential in the number of state variables. A commonly used binary method is the General Multiple-Winner (GMW) model, which can be applied to any circuit started in any state. In contrast to this, the ternary analysis method called ternary simulation is polynomial in the number of state variables, but applies only to a circuit started in a stable state. This method has been in use since 1965. The equivalence of ternary simulation to the GMW analysis was proved in 1987, for the case of a stable starting state. In this paper we present a generalized ternary simulation algorithm applicable to any state, and we prove that the new algorithm is equivalent to GMW analysis. The new algorithm is used to prove that certain behaviors are not realizable.*

1. INTRODUCTION

Asynchronous circuit theory has been developed in the 1950's [11, 12]. The interested reader is referred to [21] for details concerning the early developments in asynchronous circuit theory, and to [5, 6, 7] for additional motivation and background relevant to the present paper.

Digital circuits have been modeled by Boolean algebra since 1938 [20]; consequently, binary methods were naturally the first to be applied to asynchronous circuits. When several gates are unstable in a circuit, they are "racing" to their new states. It is normally assumed that any subset of these gates can change to their new values, *i.e.*, there can be "multiple

(*) This research was supported by the Information Technology Research Center of Ontario, by the Natural Sciences and Engineering Research Council of Canada under Grants No. OGP0000871 and OGP0109688, and by a fellowship from the Advanced Systems Institute.

([1]) Department of Computer Science University of British Columbia, Vancouver, British Columbia, Canada V6T 1Z4, email: seger@cs.ubc.ca

([2]) Department of Computer Science University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, email: brzozo@math.uwaterloo.ca

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754/94/03-04/$ 4.00/

winners" in a race. One of the central problems in asynchronous circuit theory is to determine the final "outcome" of races. The early informal binary "race analysis" model [11, 12] was formalized in 1979 [9] as the "General Multiple-Winner" (GMW) model. Here "general" refers to the fact that no assumptions are made about the gate delays, except that they are finite. A GMW analysis of a race can be exponential in the number of gates.

In 1965, Eichelberger introduced a ternary method for the analysis of races and hazards [10], building on some earlier work [23]. His algorithm, called ternary simulation, assumes a stable initial state, and is polynomial in the number of gates. Eichelberger established a connection between ternary simulation and the binary methods, but this was done informally. In 1979 Brzozowski and Yoeli [9] formulated a conjecture that ternary simulation is, in a certain sense, equivalent to GMW analysis, provided that all gate and wire delays are taken into account in the GMW model. This conjecture was settled positively by Brzozowski and Seger in 1987 [4], for the case of a stable initial state.

In this paper we present a generalized ternary simulation applicable to any state, and we prove that the new algorithm is equivalent to GMW analysis.

The paper is structured as follows. Section 2 defines the basic network model, and Section 3 describes the GMW analysis method. An introduction to ternary models is the topic of Section 4. The definition and properties of the new Algorithm A – the first of the two ternary algorithms – are then given in Section 5. The second ternary algorithm, Algorithm B, is then briefly described in Section 6; this algorithm is unchanged. Section 7 discusses some applications of the main theorem. Appendix A gives the proofs of the key results.

2. NETWORK MODELS

In this section we define a mathematical model of a gate circuit. For additional information concerning gate circuits the reader should refer to a basic text on logic design, for example [8, 13, 14, 16].

A *gate* is a physical device intended to implement a Boolean fucntion. It has $k \geq 1$ *inputs* and one *output*. If we apply binary signals at the gate inputs, the output value is determined by the gate type defined by a Boolean function. The two binary values (0 and 1) are realized by two voltage levels (low and high). In reality, a gate signal may also have an intermediate value between high and low; we then assign to this signal a third value Φ.

We now describe the structural properties of a circuit by a directed graph. A *circuit graph* is a 5-tuple $G = \langle \mathcal{X}, \mathcal{I}, \mathcal{G}, \mathcal{W}, \mathcal{E} \rangle$, where

- $\mathcal{X}$ is a set of *input vertices*, labeled $X_1, X_2, \ldots, X_n$,
- $\mathcal{I}$ is a set of *input delay vertices*, labeled $x_1, x_2, \ldots, x_n$,
- $\mathcal{G}$ is a set of *gate vertices*, labeled $y_1, y_2, \ldots, y_r$,
- $\mathcal{W}$ is a set of *wire vertices*, labeled $z_1, z_2, \ldots, z_p$, and
- $\mathcal{E} \subseteq (\mathcal{X} \cup \mathcal{I} \cup \mathcal{G} \cup \mathcal{W}) \times (\mathcal{I} \cup \mathcal{G} \cup \mathcal{W})$ is a set of *edges*.

The input vertices are all of indegree 0, and all the wire vertices have indegree and outdegree equal to 1. The directed graph defined by $((\mathcal{X} \cup \mathcal{I} \cup \mathcal{G} \cup \mathcal{W}), \mathcal{E})$ must be a bipartite graph with the vertex set separated into two disjoint subsets $\mathcal{I} \cup \mathcal{G}$ and $\mathcal{X} \cup \mathcal{W}$. Note that loops (edges of the form (v, v)) are also excluded.

Given a gate circuit, we obtain its circuit graph as follows. First, there is a vertex (called an input vertex) for every external input X_i, and a (gate) vertex for every gate. For every input vertex X_i there is an input delay vertex x_i and edge from X_i to x_i. For every input i of every gate g in the circuit there is a wire vertex z, and an edge from vertex z to the gate vertex corresponding to g. If i is connected to an external input X_j, there is an edge from the input delay vertex x_j to wire vertex z. Otherwise, if i is connected to the output of gate g', there is an edge from the gate vertex corresponding to g' to the wire vertex z.

We now turn our attention to the behavior of a circuit. The domain $\mathcal{D}$ of a circuit specifies a set of values for the circuit variables. In this paper we use either the binary domain $\{0, 1\}$, or the ternary domain $\{0, \Phi, 1\}$.

In order to describe the behavior of a vertex we associated with it a function, called the *vertex function*. For a gate vertex y_i, the vertex function Y_i maps a wire-vertex state to $\mathcal{D}$, *i.e.*, $Y_i : \mathcal{D}^{|\mathcal{W}|} \rightarrow \mathcal{D}$. This function is related to the Boolean function associated with the physical gate at that vertex. For a wire vertex z_i, the vertex function Z_i, $Z_i : \mathcal{D}^{|\mathcal{I}|+|\mathcal{G}|} \rightarrow \mathcal{D}$, provides the value of the input delay or gate vertex connected to the incoming edge of the wire vertex. For an input vertex X_i, the vertex function, also called X_i, maps a state of the environnement to $\mathcal{D}$. In contrast to the value X_i supplied by the environment, the input delay variable x_i holds the input value "seen" by the circuit.

The vertex functions defined above introduce a distinction between the present value of a vertex variable and the present value of the "excitation" of that vertex variable, *i.e.*, the value computed by the vertex function. This

permits us to associate a delay with every input, every gate, and every wire in the circuit.

In order to represent the state of the entire circuit, we need to select a set of *state variables* (or *state vertices*). We could select *all* of the vertex variables as state variables, *i.e.*, use the *input-, gate-, and wire-state* model. For some purposes, the set of state variables can be smaller; for example, a gate-state model is frequently used. For a more detailed discussion of the problem of choosing state variables see [5, 6, 7].

Assuming that the state variables are somehow selected, we now proceed to analyze the circuit using these state variables. We associate with each state vertex two distinct items: the vertex variable and its *excitation function* defined as follows. We start with the vertex function. We then repeatedly remove all dependencies on vertices which have not been chosen as state vertices, by using functional composition of the vertex functions.

We use a graph to show the functional dependencies among the state variables. This graph, called the *network*, has two sets of vertices: *input excitation vertices* and *state vertices*. There is an input excitation vertex for every external input, and a state vertex for every state variable. There is an edge from vertex i to vertex j if the excitation function of vertex j depends [1] on the variable associated with vertex i.

In summary, our formal network model has the form:

$$N = \langle \mathcal{D}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle,$$

where $\mathcal{D}$ is the domain, $\mathcal{X}$ is the set of input excitation vertices labeled $X_1, \ldots, X_n$, $\mathcal{S}$ is the set of state vertices with two sets of labels: state variable labels $(s_1, \ldots, s_m)$, and the corresponding excitation function labels $(S_1, \ldots, S_m)$, and $\mathcal{E}$ is the set of edges.

3. GENERAL MULTIPLE WINNER MODEL

A *total state* $c = a \cdot b$ of a network is an $(n + m)$-tuple of values from $\{0, 1\}$, the first n values being the input excitations, and the remaining m the variables $s_1, \ldots, s_m$. We refer to latter as (*internal*) *state* variables.

[1] We use the standard notion of functional dependence: A function f of n variables $x_1, \ldots, x_n$ depends on x_i if there exist two input n-tuples $a = (a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)$ and $a' = (a_1, \ldots, a_{i-1}, a_i', a_{i+1}, \ldots, a_n)$ such that $f(a) \neq f(a')$.

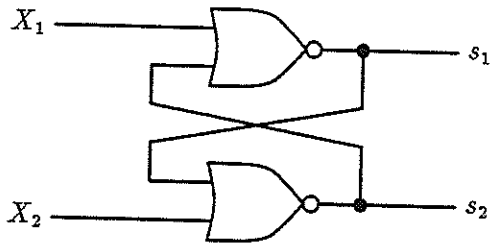

Figure 1. – NOR latch

To simplify notation, in examples we write tuples without parentheses and commas; the · is used as a separator to improve readability.

In any total state $c = a \cdot b$, the set of unstable state variables is

$$\mathcal{U}(a \cdot b) = \{s_i | b_i \neq S_i(a \cdot b)\}.$$

We next define a binary relation R_a on the set $\{0, 1\}^m$ of internal states of N for $a \in \{0, 1\}^n$:

For any $b \in \{0, 1\}^m$,

$bR_a b$, if $\mathcal{U}(a \cdot b) = \emptyset$, *i.e.*, the total state $a \cdot b$ is stable,

$bR_a b^{\mathcal{K}}$, if $\mathcal{U}(a \cdot b) \neq \emptyset$, and $\mathcal{K}$ is any nonempty subset of $\mathcal{U}(a \cdot b)$,

where by $b^{\mathcal{K}}$ we mean b with all the variables in $\mathcal{K}$ complemented. No other pairs of states are related by R_a. The relation R_a is called the *general multiple-winner* (GMW) relation [9].

We depict R_a by a directed graph, drawing an edge from b to b' if $bR_a b'$. Such an edge indicates that b' is a possible immediate successor of b. A loop from b to b indicates that the total state $a \cdot b$ is stable. The graph is a description of the possible network behaviors under the assumption that the input excitation remains constant at the value a.

To illustrate these ideas, consider the NOR latch circuit of Figure 1. If we use the gate-state network, the excitation functions are:

$$S_1 = \overline{(X_1 + s_2)} \quad \text{and} \quad S_2 = \overline{(X_2 + s_1)}.$$

The graphs of the R_a relations are shown in Figure 2.

In many applications, we are only interested in the "final outcome" of a transition, and not in the intermediate states that the network may go through before the final outcome is reached. Since every graph of $R_a(b)$ is finite, every path from b must eventually reach a cycle. A cycle in the relation

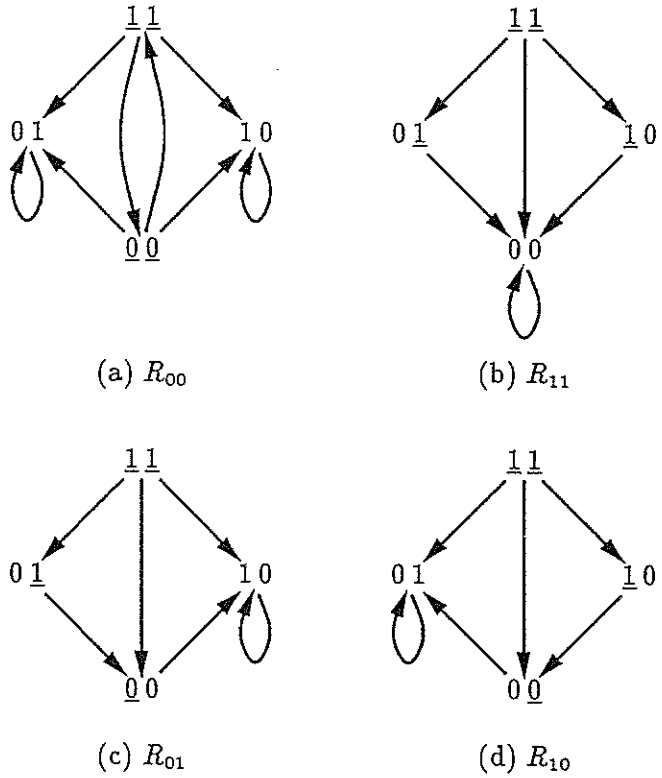

Figure 2. – R_a **relations for the NOR latch: (a)** R_{00}**, (b)** R_{11}**, (c)** R_{01}**, (d)** R_{10}**.**

diagram of R_a (b) is *transient*, if there exists a state variable s_i which has the same value in all the states of the cycle and which is unstable in each state of the cycle. If the delay of each gate is less than or equal to D, a network can stay in a transient cycle for at most D units of times. Let the set of *cyclic states* reachable from b in the relation diagram of R_a (b) be:

$$cycl\ (R_a\ (b)) = \{s \in \{0,\ 1\}^m | bR_a^* s \text{ and } sR_a^+ s\},$$

where R^+ is the transitive closure of R, and R^* is the reflexive-and-transitive closure of R. Also define the set of transient cyclic states:

$$cycl\_trans\ (R_a\ (b)) = \{s | s \text{ appears only in transient cycles}\}.$$

Next, define the set of non-transient cyclic states to be:

$$cycl\_non\_trans\ (R_a\ (b)) = \{s | s \text{ appears in a non-transient cycle}\}.$$

Now the final outcome of the transition from b is:

$$out\,(R_a\,(b)) = \{s | bR_a^* c \text{ and } cR_a^* s, \text{ where } c \in cycl\_non\_trans\,(R_a\,(b))\}.$$

Each state in *out* appears in at least one non-transient cycle, or is reachable from a state in a non-transient cycle. Informally, a state is in the outcome if the network could be found in that state at any time arbitrarily long after the start of the transition.

4. TERNARY MODELS

In analyzing the behavior of asynchronous circuits, it is often convenient to work in a ternary, rather than Boolean, algebra [8, 9, 10]. We will use the two Boolean values 0 and 1, and a third value Φ, which represents an "uncertain value", that is neither 0 nor 1. In order to improve readability, ternary variables will be set in boldface type.

We define the "uncertainty" partial order $\sqsubseteq$ on $\{0, \Phi, 1\}$ as follows:

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

and no other pairs are related by $\sqsubseteq$. Thus, for $\mathbf{s}, \mathbf{t} \in \{0, \Phi, 1\}$, the statement $\mathbf{s} \sqsubseteq \mathbf{t}$ is interpreted as $\mathbf{s}$ "has no more uncertainty" than $\mathbf{t}$. When $\mathbf{s} \sqsubseteq \mathbf{t}$, we will say that $\mathbf{s}$ is *covered* by $\mathbf{t}$ or that $\mathbf{t}$ *covers* $\mathbf{s}$. The partial order is also extended to $\{0, \Phi, 1\}^m$, for any $m > 1$, in the natural way:

$$\mathbf{s} \sqsubseteq \mathbf{t} \quad \text{iff} \quad \mathbf{s}_i \sqsubseteq \mathbf{t}_i \quad \text{for all } i, \quad 1 \leqq i \leqq m,$$

OR	0	Φ	1
0	0	Φ	1
Φ	Φ	Φ	1
1	1	1	1

AND	0	Φ	1
0	0	0	0
Φ	0	Φ	Φ
1	0	Φ	1

0	Φ	1
1	Φ	0

Figure 3. – Ternary OR, AND, and INV

where $\mathbf{s} = \mathbf{s}_1, \ldots, \mathbf{s}_m$ and $\mathbf{t} = \mathbf{t}_1, \ldots, \mathbf{t}_m$, are any two elements of $\{0, \Phi, 1\}^m$. We write $\mathbf{s} \sqsubset \mathbf{t}$ if $\mathbf{s} \sqsubseteq \mathbf{t}$ and $\mathbf{s} \neq \mathbf{t}$. Thus, for example, $0\Phi10 \sqsubset 0\Phi1\Phi$, but $0\Phi1$ and $1\Phi1$ are not related by $\sqsubseteq$.

In the partially ordered set $\{0, \Phi, 1\}^m$, we define the concept of *least upper bound* as usual. For example, $lub\,\{0, 1\} = \Phi$ and $lub\,\{\Phi010, 1110, 0100\} = \Phi\Phi\Phi0$.

For any Boolean function $f : \{0, 1\}^m \rightarrow \{0, 1\}$, its ternary extension $\mathbf{f} : \{0, \Phi, 1\}^m \rightarrow \{0, \Phi, 1\}$ is defined by:

$$\mathbf{f}(\mathbf{t}) = lub\{f(t) | t \in \{0, 1\}^m \text{ and } t \sqsubseteq \mathbf{t}\}.$$

For example, let f be the two-argument (inclusive) OR function; then

$$\mathbf{f}(0\Phi) = lub\{f(00), f(01)\} = lub\{0, 1\} = \Phi.$$

Note that any Boolean function f agrees with its ternary extension $\mathbf{f}$ when the argument $\mathbf{t}$ is binary.

The reader can verify that the functions defined in Figure 3 are the ternary extensions of the Boolean functions OR (+), AND ($\circ$) and INV ($^-$) (inversion or complement). We use the same symbols for ternary extensions of AND, OR, and INV, as we do for the binary functions.

The following important property, the *monotonicity property*, is easily verified to hold for the ternary extension $\mathbf{f}$ of any Boolean function f:

$$\mathbf{s} \sqsubseteq \mathbf{t} \text{ implies } \mathbf{f}(\mathbf{s}) \sqsubseteq \mathbf{f}(\mathbf{t}),$$

for all $\mathbf{s}, \mathbf{t} \in \{0, \Phi, 1\}^m$. This property is interpreted as follows: If input vector $\mathbf{t}$ is at least as uncertain as input vector $\mathbf{s}$, then the gate output $\mathbf{f}(\mathbf{t})$ cannot be less uncertain than $\mathbf{f}(\mathbf{s})$.

5. ALGORITHM A

In ternary simulation we use the domain $\{0, \Phi, 1\}$, and we replace the Boolean excitation functions by their ternary extensions. To distinguish two versions of the same network, one with a binary and the other with a ternary domain, we denote them by N and $\mathbf{N}$, respectively. Let $N = \langle\{0, 1\}, \mathcal{X}, \mathcal{S}, \mathcal{E}\rangle$ be a binary network, and $\mathbf{N} = \langle\{0, \Phi, 1\}, \mathcal{X}, \mathcal{S}, \mathcal{E}\rangle$ its ternary counterpart, called the *ternary extension* of N. There are n inputs and m state variables in N and $\mathbf{N}$. State variable vectors in the ternary domain are denoted by $\mathbf{s}$ and the input and vertex excitation function vectors by $\mathbf{X}$ and $\mathbf{S}$. Let $a \cdot b$ be a (binary) total state of $\mathbf{N}$. Our new first algorithm of ternary simulation is formally defined as follows:

Algorithm A

$h := 0$;

$\mathbf{s}^0 := b$;

repeat
$h := h + 1$;
$\mathrm{s}^h := lub\,\{\mathrm{s}^{h-1},\ \mathrm{S}\,(a \cdot \mathrm{s}^{h-1})\}$;
until $\mathrm{s}^h = \mathrm{s}^{h-1}$;

In the following, we use A (roman) to denote the name of the algorithm and *A* (*italic*) to denote the length of the sequence of states that the algorithm produces. Propositions 1 and 2 below are based on [9].

PROPOSITION 1: *Algorithm* A *produces a finite sequence* $\mathrm{s}^0, \ldots, \mathrm{s}^A$ *of states, where* $A \leqq m$. *Furthermore, this sequence is monotonically increasing, i.e.,*

$$\mathrm{s}^h \sqsubset \mathrm{s}^{h+1}, \qquad for \qquad 0 \leqq h < A.$$

Proof: First, by the fact that $\mathrm{t} \sqsubseteq lub\,\{\mathrm{t},\ \mathrm{t}'\}$ for any $\mathbf{t}$, $\mathbf{t}'$, it follows that

$$\mathrm{s}^h \sqsubseteq lub\,\{\mathrm{s}^h,\ \mathrm{S}\,(a \cdot \mathrm{s}^h)\} = \mathrm{s}^{h+1}, \qquad \text{for} \qquad 0 \leqq h < A.$$

Second, in each step of the algorithm, at least one state variable must become Φ; otherwise the algorithm terminates. Since there are m state variables, it follows that A cannot exceed m. □

Let N be a network in state b with inputs held constant at a. Define the set of all states reachable from b in the GMW analysis as: $reach\,(R_a\,(b)) = \{c | bR_a^* c\}$. In the following, if $h > A$, by s^h we mean s^A.

PROPOSITION 2: *The least upper bound of the set of all the states reachable in the GMW analysis of a network N is covered by the result of Algorithm* A *for* **N**, *i.e.*,

$$lub\,reach\,(R_a\,(b)) \sqsubseteq \mathrm{s}^A.$$

Moreover,

$$b\,(R_a)^h\,c\,implies\,c \sqsubseteq \mathrm{s}^h.$$

Proof: The proof of the second claim is by induction on h. For $h = 0$, we have $b\,(R_a)^0\,c$ implies $c = b$. But also $\mathrm{s}^0 = b$. Hence $b\,(R_a)^0\,c$ implies $c \sqsubseteq \mathrm{s}^0$. Assume now that $b\,(R_a)^h\,c$ implies $c \sqsubseteq \mathrm{s}^h$, and suppose that $cR_a\,d$. By definition of R_a, each component d_i of d has either the value of the corresponding component c_i in c or it is equal to the excitation $S_i\,(a \cdot c)$. Thus $d \sqsubseteq lub\,\{c,\ S\,(a \cdot c)\}$. The latter expression is equal to $lub\,\{c,\ \mathrm{S}\,(a \cdot c)\}$, since the ternary extension **S** agrees with S on binary

arguments. Using the induction hypothesis, the monotonicity of $\mathbf{S}$, and the monotonicity of *lub*, we find $d \sqsubseteq lub\,\{\mathrm{s}^h,\ \mathrm{S}\,(a \cdot \mathrm{s}^h)\} = \mathrm{s}^{h+1}$. Thus the second claim holds. By Proposition 1, s^A covers s^h for every h; hence the main claim is established. □

The main result of this section is the following theorem:

THEOREM 1: *Let $N = \langle\{0, 1\}, \mathcal{X}, \mathcal{S}, \mathcal{E}\rangle$ be an input-, gate-, and wire-state binary network, and let $\mathbf{N} = \langle\{0, \Phi, 1\}, \mathcal{X}, \mathcal{S}, \mathcal{E}\rangle$ be its ternary counterpart. If N and $\mathbf{N}$ are started in total state $a \cdot b$, then the result s^A of Algorithm* A *for $\mathbf{N}$ is equal to the lub of the set of all the states reachable from the initial state in the GMW analysis of N, i.e.,*

$$\mathrm{s}^A = lub\, reach\,(R_a\,(b)).$$

Proof: By Proposition 2, $lub\, reach\,(R_a\,(b))$ is covered by the result of Algorithm A for $\mathbf{N}$. It remains to be shown that the *lub* of the reachable states of N covers s^A. This follows from Corollary 2 in Appendix A. In the corollary it is shown that, for every vertex j, there is a state $s^j \in \{0,1\}^m$ such that $bR_a^* s^j$ and $\mathrm{s}_j^A \sqsubseteq lub\,\{b_j,\ s_j^j\}$. This is sufficient to prove the result. □

6. ALGORITHM B

For completeness, we include a description of Algorithm B which is unchanged, although the proof of Proposition 4 is modified to fit the new definition of outcome. In Algorithm B, we see how much of the uncertainty introduced by Algorithm A is eventually removed, if the network is started in the state produced by Algorithm A and the binary input a is applied.

Algorithm B

$h := 0$;
$\mathrm{t}^0 := \mathrm{s}^A$;
repeat
  $h := h + 1$;
  $\mathrm{t}^h := \mathrm{S}\,(a \cdot \mathrm{t}^{h-1})$;
until $\mathrm{t}^h = \mathrm{t}^{h-1}$;

PROPOSITION 3: *Algorithm* B *produces a finite sequence $\mathrm{t}^0, \ldots, \mathrm{t}^B$ of states, where $B \leqq m$. Furthermore, this sequence is monotonically decreasing, i.e.,*

$$\mathrm{t}^h \sqsupset \mathrm{t}^{h+1}, \qquad \text{for} \quad 0 \leqq h < B.$$

Proof: We first prove by induction on h that $\mathrm{t}^h \sqsupseteq \mathrm{t}^{h+1}$. For the basis, observe that $\mathrm{s}^A = lub\,\{\mathrm{s}^A, \mathbf{S}\,(a \cdot \mathrm{s}^A)\}$. It follows from the properties of *lub* that $\mathrm{t}^0 = \mathrm{s}^A \sqsupseteq \mathbf{S}\,(a \cdot \mathrm{s}^A) = \mathrm{t}^1$. Now assume inductively that $\mathrm{t}^h \sqsupseteq \mathrm{t}^{h+1}$. By the monotonicity of $\mathbf{S}$ if follows that

$$\mathrm{t}^{h+1} = \mathbf{S}\,(a \cdot \mathrm{t}^h) \sqsupseteq \mathbf{S}\,(a \cdot \mathrm{t}^{h+1}) = \mathrm{t}^{h+2},$$

and the induction step goes through. In view of this, at least one state variable must change from Φ to a binary value in each step of the algorithm; otherwise the algorithm terminates. Since there are m state variables, B cannot exceed m, and the proposition follows. □

PROPOSITION 4: *The least upper bound of the set of all the states in the outcome of the GMW analysis of a network N is covered by the result of Algorithm* B *for* **N**, *i.e.*,

$$lub\ out\,(R_a\,(b)) \sqsubseteq \mathrm{t}^B.$$

Moreover, for every $h \geqq 0$,

$$lub\ out\,(R_a\,(b)) \sqsubseteq \mathrm{t}^h.$$

Proof: We prove the latter claim by induction on h. If $h = 0$, then $\mathrm{t}^h = \mathrm{s}^A$. Since $out\,(R_a\,(b)) \subseteq reach\,(R_a\,(b))$, we have $lub\ out\,(R_a\,(b)) \sqsubseteq \mathrm{s}^A$ by Theorem 1, and the basis holds. Now suppose that $h > 0$ and that t^h satisfies the claim, but t^{h+1} does not. Then there must exist $c \in out\,(R_a\,(b))$ and a vertex i such that $c_i \not\sqsubseteq (\mathrm{t}^{h+1})_i$. Since $c_i \in \{0, 1\}$, this can only happen if $(\mathrm{t}^{h+1})_i = \overline{c_i}$. We now assert that the excitation $S_i\,(a \cdot d)$ is equal to $(\mathrm{t}^{h+1})_i$ for every state d in $out\,(R_a\,(b))$. Note that

$$S\,(a \cdot d) = \mathbf{S}\,(a \cdot d) \sqsubseteq \mathbf{S}\,(a \cdot \mathrm{t}^h) = \mathrm{t}^{h+1},$$

where the inequality follows from the inductive assumption (which implies $d \sqsubseteq \mathrm{t}^h$) and the monotonicity of $\mathbf{S}$. Now, since $(\mathrm{t}^{h+1})_i$ is binary and covers $S_i\,(a \cdot d)$, it must be equal to $S_i\,(a \cdot d)$, as claimed. Now consider any non-transient cycle in $out\,(R_a\,(b))$. Since the excitation of the i-th variable is constant throughout the cycle, the value of the variable must be constant throughout the cycle. Since the cycle is non-transient, that value must be equal to the excitation. Thus $d_i = (\mathrm{t}^{h+1})_i = \overline{c_i}$ for every state d in the cycle. Since the non-transient cycle was arbitrary, we have shown that $d_i = (\mathrm{t}^{h+1})_i = \overline{c_i}$ for every state d in every non-transient cycle in $out\,(R_a\,(b))$. This, together with the fact that $S_i\,(a \cdot e) = \overline{c_i}$ for every state e

in *out* $(R_a(b))$, implies that every state $d \in$ *out* $(R_a(b))$ reachable from a non-transient cycle will also have $d_i = (\mathrm{t}^{h+1})_i = \overline{c_i}$. However, these results together imply that $d_i = \overline{c_i}$ for every $d \in$ *out* $(R_a(b))$, contradicting the assumption that $c \in$ *out* $(R_a(b))$. Hence, the induction step goes through. The main claim of the proposition now follows in view of Proposition 3. □

The characterization of the results of Algorithm B is given in the following:

THEOREM 2: *Let* $N = \langle \{0, 1\}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$ *be an input-, gate-, and wire-state binary network, and let* $\mathbf{N} = \langle \{0, \Phi, 1\}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$ *be its ternary counterpart. If* N *and* $\mathbf{N}$ *are started in total sate* $a \cdot b$, *then the result* t^B *of Algorithm* B *is equal to the lub of the outcome of the GMW analysis, i.e.,*

$$\mathrm{t}^B = lub\ out\ (R_a(b)).$$

Proof: By Proposition 4, *lub out* $(R_a(b))$ is covered by the result of Algorithm B. It remains to be shown that the *lub* of all the states in the outcome of N covers t^B. This follows from Lemma 9 in Appendix A. In the lemma it is shown that there exists a non-transient cycle Z reachable from the initial state and such that the *lub* of all the states in Z covers t^B. This suffices to prove the theorem. □

An important corollary that follows directly from the construction in the proof is:

COROLLARY 1: *Let* N *and* $\mathbf{N}$ *be as in Theorem 2, and let* t^B *be the result of Algorithm* B *when* $\mathbf{N}$ *is started in total state* $a \cdot b$. *If* t^B *is not binary, then there is a non-transient cycle, reachable from* $a \cdot b$ *in the GMW analysis of* N, *such that every gate and wire vertex* j *with* $\mathrm{t}_j^B = \Phi$ *oscillates.*

7. DELAY-INSENSITIVE CIRCUITS

A delay-insensitive circuit functions correctly independently of the sizes of the delays in its components and wires. Consequently, the verification of such a circuit by a GMW analysis requires an input-, gate-, and wire-state network model. In this section we show that our new ternary simulation can be a powerful tool for proving that certain behaviors cannot be realized delay-insensitively.

Traditionally [11, 12, 15, 21], asynchronous circuits have been operated in *fundamental mode*, in which the environment is allowed to change the circuit inputs only if the circuit is stable. More recent asynchronous design techniques use the *input/output mode* of operation [2, 3, 17, 22]. In this

mode, the environment does not have to wait until the circuit has stabilized completely; a new input can be applied as soon as the circuit has given an appropriate output response. The analysis of circuits operated in the input/output mode requires the more general form of ternary simulation developed in this paper. The work below follows closely the ideas of [3]. Lemma 1 is a generalization of the result of [3], where a more restricted definition of input/ouput mode-realization was used along with the original version of Algorithm A.

In order to show that certain behaviors cannot be implemented delay-insensitively by gate circuits, we first prove that a particular very simple behavior, called A_1, cannot be implemented. We then use reduction techniques to show that several common behaviors, like that of the C-ELEMENT [19] and the set-reset latch, cannot be implemented.

The behavior A_1 is defined as follows. It has one input X and one output O, and it is operated in the input/output mode. The initial "input/output state" of the behavior is $X \cdot O = 0 \cdot 0$, and this state is stable, in the sense that the output will not change unless the input changes. Once the input has changed, the behavior reaches the state $1 \cdot 0$; this state is unstable, because the output should (eventually) change to 1, resulting in state $1 \cdot 1$. As soon as the output has changed, the environment is allowed to change the input back to 0. However, the behavior should not change the output again, *i.e.*, it must remain in state $0 \cdot 1$. This can be summarized by the following transitions:

$$0 \cdot 0 \xrightarrow{\{X\}} 1 \cdot 0 \xrightarrow{\{O\}} 1 \cdot 1 \xrightarrow{\{X\}} 0 \cdot 1.$$

Any network N realizing A_1 must have the following properties:

P_1 If $q_1 = 0 \cdot b$ is a state of N representing the initial state of the behavior, then every state c (including b) reachable by an R_0-sequence from b must have the output O equal to 0. (The output cannot change by itself.)

P_2 The input is allowed to change in any state $0 \cdot c$, defined as above, and the state $1 \cdot d$ reached after this input change must be unstable and must have $O = 0$. (The output cannot change instantly, because the output wire has a delay.)

P_3 In every R_1-sequence starting with d and ending with a state in *out* (R_1, d), O changes exactly once. (Exactly one output change is specified in A_1.)

P_4 Let e be any state that can be reached by an R_1-sequence from d and that has $O = 1$. Then the input is allowed to change again. The state

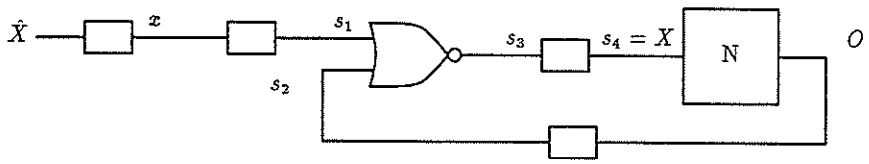

Figure 4. – Network $\tilde{N}$

$0 \cdot f$ so reached must have $O = 1$. (Input-output mode permits an input change as soon as the output changes.)

P_5 Every state g reached from f by an R_0-sequence must have $O = 1$. (The output should not change again.)

We now show that no such delay-insensitive design exists.

LEMMA 1: *The behavior A_1 does not have a delay-insensitive input/output-mode realization.*

Proof: We show that, if a network N with initial state q_1 that is a delay-insensitive input/output-mode realization of A_1 existed, then we could construct a network $\tilde{N}$ that would have contradictory properties.

Consider the network $\tilde{N}$ derived from N as shown in Figure 4. Notice that a delay element is introduced for the input as well as every wire. Since, by assumption, network N also contains a delay element for each wire, we have an input-, gate- and wire-state network model for $\tilde{N}$. Let s' denote the vector of internal state variables of N, except for the output variable which is denoted by O.

The initial state of $\tilde{N}$ is $\tilde{X} \cdot xss'O = \tilde{X} \cdot xs_1s_2s_3Xs'O = 0 \cdot 11000b0$. Consider now any $\tilde{R}_0$-sequence. Note that, by conditions P_1 and P_2, the output O of N will not change before X changes to 1. Note also that, eventually, x, s_1, s_3, and X will change because the input delay is unstable. In fact, every $\tilde{R}_0$-sequence starting in $0 \cdot 11000b0$ can lead to any state of the form $00011d0$, where d is reachable form c in N by an R_0-sequence. Because of Property P_2, all such states must be unstable. By P_3, N eventually reaches a state $1 \cdot e1$, for some vector e. Thus we must have an $\tilde{R}_0$-sequence $00011d0 \rightarrow^* 00011e1$. From P_4 and P_5 it now follows that O cannot change any more, even if X becomes 0 again; this has to hold for all possible values that s' may reach. Thus, the s'-component of the state of $\tilde{N}$ becomes irrelevant, and we replace it by # from now on. After O becomes 1, we have the following $\tilde{R}_0$-sequence:

$$00011e1 \rightarrow 00111\#1 \rightarrow 00101\#1 \rightarrow 00100\#1.$$

In the last state, the variables x, s_1, s_2, s_3, X, and O are stable and will not become unstable again. It follows that the outcome of the GMW analysis of $\tilde{N}$ started in state $0 \cdot 11000b0$, always yields states of the form $0 \cdot 00100\#1$, *i.e.*,

$$h \in out\,(\tilde{R}_0,\ 11000b0) \text{ implies the } O \text{ component of } h \text{ is } 1.$$

Consequently, even in the presence of arbitrary input, gate and wire delays, the final outcome of the transition yields $O = 1$. Note that, in the analysis above, N is operated in input/output mode.

Next we show that ternary simulation of $\tilde{N}$ contradicts the conclusion reached above. By Property P_1, as long as the input X of N is 0, the excitation of the output delay must be 0. Hence the output delay is initially stable. Algorithm A produces the following sequence:

$$0 \cdot 11000b0 \rightarrow 0 \cdot \Phi 1000\#0 \rightarrow 0 \cdot \Phi\Phi 000\#0 \rightarrow 0 \cdot \Phi\Phi 0\Phi 0\#0 \rightarrow 0 \cdot \Phi\Phi 0\Phi\Phi\#0,$$

where the # indicates that we don't know the values of the s' portion of the state. Trivially, $11000b0\ \tilde{R}_0^*\ 11000b0$, and we have shown above that $11000b0\ \tilde{R}_0^*\ 00011e1$, *i.e.*, both $11000b0$ and $00011e1$ are reachable from $11000b0$ (in zero or more steps). Consequently, the output O can take the values 0 and 1 in the GMW analysis of the network. By Proposition 2, Algorithm A must produce $O = \Phi$. Subsequently, s_2 becomes Φ, and the final result of Algorithm A has the form $0 \cdot \Phi\Phi\Phi\Phi\Phi\mathrm{t}\Phi$ for some vector t of ternary values.

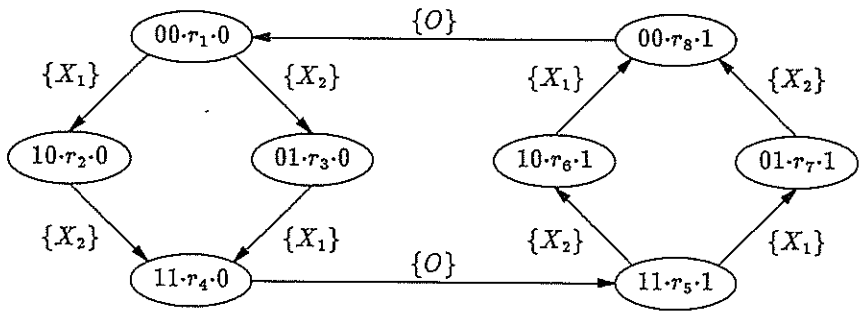

Figure 5. – Behavior of C-ELEMENT

Applying Algorithm B to state $0 \cdot \Phi\Phi\Phi\Phi\Phi\mathrm{t}\Phi$, we find that it terminates in the third step with state $0 \cdot 00\Phi\Phi\Phi\mathrm{t}\Phi$. Consequently, Algorithm B predicts

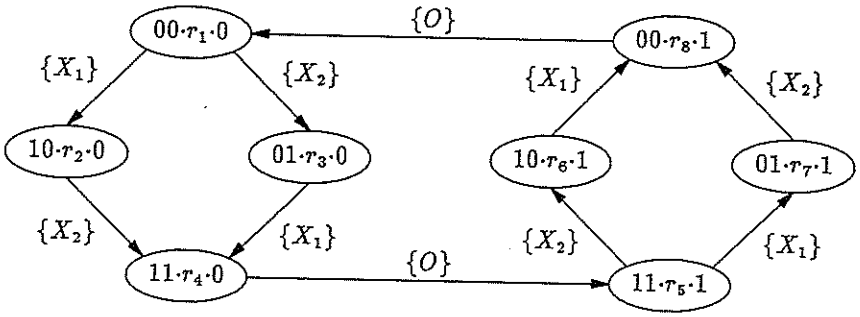

Figure 5. – Behavior of C-ELEMENT

that O has the value Φ. But then, by Theorem 2, there exists a state in the outcome of the GMW analysis where $O = 0$. This contradicts the GMW analysis above. Therefore, the network N with the postulated properties cannot exist, and we have proved that behavior A_1 does not have a delay-insensitive gate realization operated in the input/output mode. □

Using Lemma 1 we use the arguments of [3] to show that the behavior of the C-ELEMENT – a basic component of delay-insensitive design – does not have a delay-insensitive realization in input/output mode. The behavior of the C-ELEMENT, with inputs X_1 and X_2 and output O, is shown in Figure 5, where the entries within the nodes consist of input, internal, and output state. Whenever the two inputs agree, the output should take on their common value. When the inputs disagree, the output should retain its current value. The C-ELEMENT has the sub-behavior:

$$10 \cdot r_2 \cdot 0 \xrightarrow{\{X_2\}} 11 \cdot r_4 \cdot 0 \xrightarrow{\{O\}} 11 \cdot r_5 \cdot 1 \xrightarrow{\{X_2\}} 10 \cdot r_6 \cdot 1.$$

If we ignore the input X_1 and associate X_2 with X, we obtain a behavior isomorphic to A_1. Thus if a delay-insensitive implementation of a C-ELEMENT operated in input/output mode existed, then so would a delay-insensitive implementation of behavior A_1 operated in input/output mode. In view of Lemma 1, the C-ELEMENT cannot be realized by a delay-insensitive gate circuit. For additional results concerning delay-insensitive realizations see [3, 7].

We conclude this section with a sketch of a proof that it is impossible to construct a delay-insensitive gate circuit that would act as an arbiter. This result was proved by [1] in a totally different formalism. A primitive arbiter has two inputs X_1 and X_2. A 1 on either input represents a request for a shared resource. The outputs O_1 and O_2 represent grants for use of the

resource. The essential function in the arbitration process is to ensure that, when X_1 and X_2 are both 1, only one of them can be served. The arbiter can grant the resource to X_1 by setting $O_1 = 1$, $O_2 = 0$, or to X_2 by setting $O_1 = 0$, $O_2 = 1$. The arbiter is not allowed to always give preference to one of its inputs but must, in fact, implement the "critical" race above. In view of Theorem 2, ternary simulation for this situation must yield $O_1 = O_2 = \Phi$. By Corollary 1, however, it is possible for O_1 and O_2 to oscillate under suitable delay assumptions. Clearly such a behavior is not allowed for an arbiter. Consequently, no delay-insensitive gate circuit can realize an arbiter. For additional details the reader is referred to [7].

Acknowledgement. – The authors would like to thank Jo Ebergen of the University of Waterloo for the new definition of outcome used in this paper and for many useful suggestions concerning this work.

A. PROOFS OF MAIN RESULTS

Let N and $\mathbf{N}$ be input-, gate-, and wire-state networks as in Theorem 1. We require some notation for referring to the fan-in and fan-out vertices of a given gate vertex i in a network. Let the set of its fan-in vertices be

$$\alpha^i = \{j | (j, i) \in \mathcal{E}\}.$$

Note that $\alpha^i \cap \alpha^j = \emptyset$ if $i \neq j$. With a slight abuse of notation, given a vector v of length m and $\alpha^i \neq \emptyset$, we write $\alpha^i(v)$ to denote the components of the vector corresponding to the fan-in vertices; thus, if $\alpha^i = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, then $\alpha^i(v) = v_{\alpha_1}, v_{\alpha_2}, \ldots, v_{\alpha_r}$. Similarly, let the fan-out vertices of i be

$$\beta^i = \{j | (i, j) \in \mathcal{E}\}.$$

Again, note that $\beta^i \cap \beta^j = \emptyset$ if $i \neq j$. Given a vector v of length m and $\beta^i \neq \emptyset$, by $\beta^i(v)$ we denote the components of the vector corresponding to the fan-out vertices of vertex i. Finally, if $\mathbf{s}^h$, $0 \leqq h \leqq A$ denotes the result of Algorithm A after h steps, and vertex $\mathbf{s}_j^A = \Phi$, let γ_j denote the step in which this vertex changes to Φ, *i.e.*, if $\mathbf{s}_j^{k-1} = b_j$ and $\mathbf{s}_j^k = \Phi$, then $\gamma_j = k$.

The following technical result concerning binary and ternary excitations will be needed in Lemma 2.

PROPOSITION 5: *Let N and $\mathbf{N}$ be as in Theorem 1, and let j be a gate vertex of N with indegree d_j and fan-in set α^j. If $\mathbf{s} \in \{0, \Phi, 1\}^m$ is such that $\mathbf{s}_j = b_j \in \{0, 1\}$ and $lub\{\mathbf{s}_j, \mathbf{S}_j(a \cdot \mathbf{s})\} = \Phi$, there exists $c^j \in \{0, 1\}^{d_j}$*

such that $c^j \sqsubseteq \alpha^j$ (s) *and* $S_j\ (a \cdot s) = \overline{b_j}$ *for every* $s \in \{0, 1\}^m$ *such that* $\alpha^j\ (s) = c^j$.

Proof: We prove the claim by contradiction. Assume that, for all $c^j \in \{0, 1\}^{d_j}$ such that $c^j \sqsubseteq \alpha^j$ (s), there is some state $s \in \{0, 1\}^m$ such that $c^j = \alpha^j\ (s)$ and $S_j\ (a \cdot s) = b_j$. Since S_j depends only on the vertices in α^j, we can conclude that $S_j = (a \cdot s) = b_j$ implies $S_j = (a \cdot s') = b_j$ for every $s' \in \{0, 1\}^m$ such that $\alpha^j\ (s) = \alpha^j\ (s')$. Altogether, we have that $S_j = (a \cdot s) = b_j$ for every $s \in \{0, 1\}^m$ such that $\alpha^j\ (s) \sqsubseteq \alpha^j$ (s). By the definition of ternary extension, this implies that S_j (s) $= b_j$; hence $lub\ \{\mathrm{s}_j,\ \mathrm{S}_j\ (a \cdot \mathrm{s})\} = lub\ \{b_j,\ b_j\} = b_j$, contradicting the assumption that $lub\ \{\mathrm{s}_j,\ \mathrm{S}_j\ (a \cdot \mathrm{s})\} = \Phi$. □

The following is the key lemma required to show that the result of Algorithm A is covered by the set of states reachable in the GMW analysis.

LEMMA 2: *Let* N *and* $\mathbf{N}$ *be as in Theorem 1, and let* s^h, $0 \leqq h \leqq A$ *be the result of Algorithm* A *after* h *steps. Then, for each* h, *there exists* $s^h \in \{0, 1\}^m$ *with the following properties:*

1. $bR_a^*\ s^h$.
2. *If* j *is an input delay or gate vertex, then*

$$s_j^h = \begin{cases} b_j & if \quad \mathrm{s}_j^h = b_j, \\ \overline{b_j} & if \quad \mathrm{s}_j^h = \Phi. \end{cases}$$

For the next two properties, let j *be a wire vertex in the fan-out set of vertex* i *and in the fan-in set of vertex* k ($i = k$ *is possible*).

3. *If* $\mathrm{s}_k^h = b_k \in \{0, 1\}$, *then*

$$s_j^h = \begin{cases} b_j & if \quad \mathrm{s}_i^h = b_i \quad and \quad \mathrm{s}_j^h = b_j, \\ \overline{S_j\ (a \cdot s^h)} & if \quad \mathrm{s}_i^h = \Phi \quad or \quad \mathrm{s}_j^h = \Phi. \end{cases}$$

4. *If* $\mathrm{s}_i^h = \mathrm{s}_k^h = \Phi$, *then*

$$\gamma_i \geqq \gamma_k \ implies\ s_j^h = \overline{S_j\ (a \cdot s^h)}.$$

Proof: We prove the lemma by induction on h. The basis, $h = 0$, follows trivially since $\mathrm{s}^0 = b \in \{0, 1\}^m$. Assume inductively that the state s^h has been reached and that s^h satisfies Properties 1-4. We will show how to reach a state s^{h+1} that satisfies all four properties. We do this in two steps. We first show that there is a state $\tilde{s}^{h+1}$ reachable from s^h in which all input delay and

gate vertices that change to Φ in step $h+1$ in Algorithm A are unstable. We then conclude the proof by showing how s^{h+1} can be reached from $\tilde{s}^{h+1}$.

It is convenient to introduce the following shorthand. Let C^{h+1} be the set of gate and input delay vertices that change to Φ in step $h+1$, *i.e.*,

$$C^{h+1} = \{j \in \mathcal{I} \cup \mathcal{G} | \mathrm{s}_j^h = b_j \quad \text{and} \quad \mathrm{s}_j^{h+1} = \Phi\}.$$

Now, let $\tilde{s}_j^{h+1} = s_j^h$ for every input delay and gate vertex. If $j \in C^{h+1}$, let $\alpha^j(\tilde{s}^{h+1}) = c^j$, where $c^j \in \{0, 1\}^{d_j}$ is such that $c^j \sqsubseteq \alpha^j(\mathrm{s}^h)$ and $S_j(a \cdot s) = \overline{b_j}$ for all $s \in \{0, 1\}^m$ such that $\alpha^j(s) = c^j$. By Proposition 5 such a c^j is guaranteed to exist. If $j \notin C^{h+1}$, let $\alpha^j(\tilde{s}^{h+1}) = \alpha^j(s^h)$. Note that this completely determines $\tilde{s}^{h+1}$. We first claim that every vertex in C^{h+1} is unstable in $\tilde{s}^{h+1}$. To verify this, consider two cases. First, if j is an input delay vertex, then h must be 0 and the input delay excitation function X_j must be $\overline{b_j}$. Thus input delay vertex j is unstable at $h = 0$. Since, by construction, no input delay vertex changes in going from s^h to $\tilde{s}^{h+1}$, input delay vertex j must still be unstable in $\tilde{s}^{h+1}$. On the other hand, if j is a gate vertex, then $\alpha^j(\tilde{s}^{h+1}) = c^j$. But c^j was chosen so that $S_j(a \cdot \tilde{s}^{h+1}) = \overline{b_j}$. By Property 2 of the induction hypothesis, $s_j^h = b_j$; thus vertex j is unstable in $\tilde{s}^{h+1}$.

We now claim that $s^h R_a^* \tilde{s}^{h+1}$. Clearly, the claim holds if $\tilde{s}^{h+1} = s^h$. Hence, assume $\tilde{s}^{h+1} \neq s^h$. It is sufficient to show that each vertex that changes in going from s^h to $\tilde{s}^{h+1}$ is unstable in the total state $a \cdot s^h$. Let j be such a vertex, *i.e.*, assume $\tilde{s}_j^{h+1} \neq s_j^h$. By construction, it follows that j must be a wire vertex in the fan-in set of some vertex $k \in C^{h+1}$ and in the fan-out set of some vertex i. However, $\alpha^k(\tilde{s}^{h+1}) = c^k$ and, by definition, $c^k \sqsubseteq \alpha^k(\mathrm{s}^h)$. In particular, $c_j^k = \tilde{s}_j{}^{h+1} \sqsubseteq \mathrm{s}_j^h$. If $\mathrm{s}_i^h = b_i$ and $\mathrm{s}_j^h = b_j$ then, by Property 3 of the induction hypothesis, $s_j^h = b_j$. However, $\mathrm{s}_j^h = b_j$ implies that $\tilde{s}_j^{h+1} = b_j$ and thus in this case $\tilde{s}_j^{h+1} = s_j^h$. On the other hand, if $\mathrm{s}_i^h = \Phi$ or $\mathrm{s}_j^h = \Phi$ then, again by Property 3 of the induction hypothesis, we have $s_j^h = \overline{S_j(a \cdot s^h)}$, and thus vertex j is unstable in the total state $a \cdot s^h$. Altogether, $\tilde{s}_j^{h+1}$ is either equal to s_j^h or $S_j(a \cdot s^h)$ for $1 \leq i \leq m$; thus $s^h R_a^* \tilde{s}^{h+1}$.

We are now ready to construct s^{h+1}. If $j \in C^{h+1}$ let $s_j^{h+1} = S_j(a \cdot \tilde{s}^{h+1})$ and $\beta^j(s^{h+1}) = \beta^j(S(a \cdot \tilde{s}^{h+1}))$. If $j \notin C^{h+1}$, let $s_j^{h+1} = \tilde{s}_j^{h+1}$ and $\beta^j(s^{h+1}) = \beta^j(\tilde{s}^{h+1})$. Note that this uniquely determines s^{h+1}. We now must verify that s^{h+1} satisfies Properties 1-4. First, it follows immediately from the construction that $\tilde{s}^{h+1} R_a^* s^{h+1}$. From the fact that $s^h R_a^* \tilde{s}^{h+1}$ and

from the induction hypothesis, it follows that $bR_a^* s^{h+1}$ and Property 1 holds. Secondly, by construction, $\tilde{s}_j^{h+1} = s_j^h$ for every input delay and gate vertex and the only gate and input delay vertices that are changed in going from $\tilde{s}^{h+1}$ to s^{h+1} are those that change to Φ at step $h+1$ in Algorithm A; hence it follows from the induction hypothesis that Property 2 holds for every gate and input delay vertex in s^{h+1}.

Now, consider any wire vertex j which is in the fan-out set of vertex i and in the fan-in set of vertex k and for which $\mathrm{s}_k^{h+1} = b_k$. If $\mathrm{s}_i^{h+1} = \mathrm{s}_i^h$ and $\mathrm{s}_j^{h+1} = \mathrm{s}_j^h$ then, by the construction, $s_i^{h+1} = s_i^h$ and $s_j^{h+1} = s_j^h$. Thus, by the induction hypothesis, Property 3 holds for j. On the other hand, if $i \in C^{h+1}$, the construction of s^{h+1} ensures that every wire vertex in the fan-out set of gate i will be unstable, since we simultaneously set its output to its current excitation and change its input. Hence, Property 3 holds in this case too. Finally, if $\mathrm{s}_i^{h+1} = b_i$ but $\mathrm{s}_j^{h+1} = \Phi$, it follows immediately that h must be 0 and that the circuit was started in a state in which wire vertex j was unstable, *i.e.*, $b_j = \overline{S_j\,(a \cdot b)}$. Since neither i nor k is in C^1, it follows that $s_i^1 = s_i^0 = b_i$ and that $s_j^1 = s_j^0 = b_j$. Since the excitation of wire vertex j is completely determined by the value on gate or input delay vertex i, it follows that wire vertex j will remain unstable in total state $a \cdot s^{h+1}$ and Property 3 holds.

Finally, consider any wire vertex j such that $j \in \beta^i$, $j \in \alpha^k$, $\mathrm{s}_i^{h+1} = \mathrm{s}_k^{h+1} = \Phi$ and $\gamma_i \geqq \gamma_k$. There are two cases to consider. If $i \in C^{h+1}$ then, by the construction of s^{h+1}, we have $s_j^{h+1} = \overline{S_j\,(a \cdot s^{h+1})}$. On the other hand, if $i \notin C^{h+1}$, then $k \notin C^{h+1}$, since otherwise $\gamma_k > \gamma_i$. However, if neither i nor k is in C^{h+1} then none of i, j, and k changes in going from s^h to s^{h+1}. Since j is a wire vertex, its excitation depends only on the value on vertex i. Consequently, the excitation of vertex j does not change in going from s^h to s^{h+1}. By Property 4 of the induction hypothesis, it follows that $s_j^{h+1} = \overline{S_j\,(a \cdot s^{h+1})}$. □

From this result, we immediately obtain the following:

COROLLARY 2: *Let N and $\mathbf{N}$ be as in Theorem 1. Then, for $1 \leqq j \leqq m$, there exists a state $s^j \in \{0, 1\}^m$ such that $bR_a^* s^j$ and*

$$lub\,\{b_j, s_j^j\} \sqsupseteq \mathrm{s}_j^A.$$

Proof: If $\mathrm{s}_j^A = b_j$, the result follows trivially. So assume $\mathrm{s}_j^A = \Phi$. If j is an input delay or gate vertex, then the result follows immediately from Lemma 2, Property 2. So assume j is a wire vertex between vertices i and k,

i.e., $(i, j) \in \mathcal{E}$ and $(j, k) \in \mathcal{E}$. If vertex j is unstable in the total state $a \cdot b$, then we can reach a state in which $s_j = \overline{b_j}$. Hence, assume wire vertex j is stable in state $a \cdot b$. The excitation of wire vertex j is completely determined by the value on vertex i; thus $S_j\,(a \cdot s) = b_i$ for every $s \in \{0, 1\}^m$ such that $s_i = b_i$. Assume vertex j changes to Φ at step r in Algorithm A. This implies that vertex i must have changed to Φ in step $r - 1$, and thus $\mathrm{s}_i^A = \Phi$. By Property 2 of Lemma 2, this implies that we can reach a state $s \in \{0, 1\}^m$ such that $s_i = \overline{b_i}$. This means that $S_j\,(a \cdot s) = \overline{b_j}$; thus we can reach a state $\tilde{s}$ in which $\tilde{s}_j = \overline{b_j}$. □

The proofs of the results below follow closely the general pattern used in [4]. The main difference is that in [4] Algorithm B was applied to a gate-state network, whereas here we apply it to an input-, gate-, and wire-state network.

Given the result t^B of Algorithm B, if $\mathrm{t}_j^B = \Phi$, we say that vertex j is *indefinite*; otherwise it is *definite*. Note that every input delay vertex is definite since we assume that the inputs to the circuit are always binary. Let $\mathcal{D}$ denote the set of definite vertices and $\mathcal{J}$ the set of indefinite vertices.

Assuming there is at least one indefinite vertex j (*i.e.*, Algorithm B does not yield a binary result), there must be some other vertex $i \in \alpha^j$ which is also indefinite. Otherwise, all inputs to vertex j would be binary and its excitation could not be Φ. Since the network **N** is finite, we must have at least one cycle of indefinite vertices; such a cycle will be called *indefinite*. Note that, since we are using an input-, gate-, and wire-state network – thus every loop in the network is of length at least two – there must be at least one gate vertex and one wire vertex in every indefinite cycle.

Eventually we want to show that, if the result of Algorithm B contains at least one Φ, there exists a non-transient cycle of length ≥ 2 (*i.e.*, an oscillation) in the graph of the relation R_a for N such that all indefinite vertices "take part" in the oscillation, *i.e.*, each vertex variable takes on both values 0 and 1 in the cycle. Furthermore, that cycle is reachable from the initial state of N.

The following definitions help to simplify the proofs. A total state $a \cdot c$ of N is *compatible* with $a \cdot \mathrm{t}^B$ if $c \sqsubseteq \mathrm{t}^B$. Also, a total state $a \cdot c$ of N is *definite stable* if all the definite vertices are stable in that state. Finally, a total state $a \cdot c$ of N is *loop unstable* if there is at least one unstable wire vertex in each indefinite cycle of N.

LEMMA 3: *Let N and* **N** *be as in Theorem 1 and let $s^A \in \{0, 1\}^m$ be a state derived as in the proof of Lemma 2. If t^h is the result of Algorithm* B

after h steps, $0 \leqq h \leqq B$, then there is a state $t^h \in \{0, 1\}^m$ such that:

I. $bR_a^* t^h$,

II. $t^h \sqsubseteq \mathrm{t}^h$,

III. *if* $\mathrm{t}_j^h = \Phi$ *then* $t_j^h = s_j^A$.

Proof: We proceed by induction on h. For the basis, let $t^0 = s^A$. Properties I-III follow trivially from the fact that $\mathrm{t}^0 = \mathrm{s}^A$, from Proposition 2, and from the assumption that s^A satisfies Properties 1-3 of Lemma 2.

Assume inductively that t^h has been constructed and let

$$t_j^{h+1} = \begin{cases} S_j\,(a \cdot t^h) & \text{if} \quad \mathrm{t}_j^{h+1} \in \{0, 1\} \\ t_j^h & \text{otherwise.} \end{cases}$$

Clearly, $t^h R_a^* t^{h+1}$. Together with the induction hypothesis Property I, it follows that $bR_a^* t^{h+1}$. For Property II, consider any vertex j. If $\mathrm{t}_j^{h+1} = \Phi$ then it follows trivially that $t_j^{h+1} \sqsubseteq \mathrm{t}_j^{h+1}$. Hence assume that $\mathrm{t}_j^{h+1} \in \{0, 1\}$. By definition of Algorithm B, $\mathrm{t}_j^{h+1} = \mathrm{S}_j\,(a \cdot \mathrm{t}^h)$. By the induction hypothesis Property II and the monotonicity of S, it follows that $\mathrm{S}_j\,(a \cdot t^h) \sqsubseteq \mathrm{S}_j\,(a \cdot \mathrm{t}^h) = \mathrm{t}_j^{h+1}$. But $\mathrm{S}_j\,(a \cdot t^h) = S_j\,(a \cdot t^h)$, since the ternary extension S agrees with S on binary arguments. By construction, $\mathrm{t}_j^{h+1} \in \{0, 1\}$ implies that $t_j^{h+1} = S_j\,(a \cdot t^h)$. Thus, it follows that $t_j^{h+1} \sqsubseteq \mathrm{t}_j^{h+1}$. Since j was arbitrary, Property II follows. Finally, by the monotonicity of Algorithm B (Proposition 3) if $\mathrm{t}_j^{h+1} = \Phi$ then $\mathrm{t}_j^h = \Phi$. However, by construction, if $\mathrm{t}_j^{h+1} = \Phi$ then $t_j^{h+1} = t_j^h$. This, together with the induction hypothesis Property III, implies that if $\mathrm{t}_j^{h+1} = \Phi$ then $t_j^{h+1} = t_j^h = s_j^A$ and Property III follows. Since Properties I-III hold for t^{h+1}, the induction goes through and the lemma follows. □

LEMMA 4: *Let $t \in \{0, 1\}^m$ be any state such that $t \sqsubseteq \mathrm{t}^B$. Then t is definite stable.*

Proof: By Proposition 3, $\mathrm{t}^B = \mathrm{S}\,(a \cdot \mathrm{t}^B)$. Now consider any definite vertex j. By the definition it follows that $\mathrm{t}_j^B \in \{0, 1\}$ and thus $\mathrm{t}_j^B = \mathrm{S}_j\,(a \cdot \mathrm{t}^B) \in \{0, 1\}$. However, by the assumption that $t \sqsubseteq \mathrm{t}^B$, and by the monotonicity of S, it follows that $\mathrm{S}_j\,(a \cdot t) \sqsubseteq \mathrm{S}_j\,(a \cdot \mathrm{t}^B) = \mathrm{t}_j^B \in \{0, 1\}$ and therefore $\mathrm{S}_j\,(a \cdot t) = \mathrm{t}_j^B$. But $\mathrm{S}_j\,(a \cdot t) = S_j\,(a \cdot t)$, since the ternary extension S agrees with S on binary arguments. Altogether, if $\mathrm{t}_j^B \in \{0, 1\}$ then $S_j\,(a \cdot t) = \mathrm{t}_j^B = t_j^B$, where the last equality follows from the fact that $t_j^B \sqsubseteq \mathrm{t}_j^B$. □

COROLLARY 3: *Let t^B be a state derived as in the proof of Lemma 3. Then t^B is definite stable.*

Proof: The proof follows immediately from Lemma 4 and Property II of Lemma 3. □

LEMMA 5: *Let t^B be a state derived as in the proof of Lemma 3. Then t^B is loop unstable.*

Proof: It is sufficient to prove the claim for each indefinite simple cycle, where a cycle is simple if it has no repeated vertices except for the first and the last vertex in the cycle. Let C be an arbitrary indefinite simple cycle in N. Note that C contains only gate and wire vertices, since no input delay vertex can be indefinite. A gate vertex i in C is said to be *terminating* if no other gate vertex in C becomes Φ in Algorithm A after vertex i. Clearly, there must be at least one terminating vertex in C. Assume vertex i is terminating in C and that it became Φ at step r of Algorithm A. Since i is in C, one of the wire vertices in β^i must be the successor vertex to i in C; assume this is vertex j. We now claim that j is unstable in t^B. Note first that since i and j are indefinite vertices, *i.e.*, $\mathrm{t}_i^B = \mathrm{t}_j^B = \Phi$ by Property III of Lemma 3, we can conclude that $t_i^B = s_i^A$ and $t_j^B = s_j^A$. Furthermore, since j is a wire vertex, its excitation is completely determined by the value on gate vertex i. Thus, if j is unstable in s^A, then it is also unstable in t^B. Finally, since i is terminating, it follows that $\gamma_i \geqq \gamma_k$ for every other gate vertex in C. In particular, if $j \in \alpha^k$ ($k = i$ is possible), then $\gamma_i \geqq \gamma_k$. By Lemma 2 Property 4 it follows that $s_j^A = \overline{S_j\,(a \cdot s^A)}$. □

The proof now proceeds as follows. Starting with a state $s \in \{0, 1\}^m$ we first exhibit a sequence of states

$$s = s^0, \quad s^1, \ldots, s^r,$$

where r is the number of indefinite gate vertices, and, for $0 \leqq k \leqq r$, exactly k indefinite gate vertices in s^k have values complementary to those in s, and the other indefinite gate vertices are the same as in s. Note that we do not say anything about the indefinite wire vertices. For convenience, we will say that k indefinite vertices have been "marked" in this way. By repeating this process of marking (*i.e.*, complementing) all of the indefinite gate vertices, we show the existence of an oscillation involving all the indefinite gate vertices. We then show that every indefinite wire vertex also oscillates in the constructed cycle.

LEMMA 6: *Let* t^B *be the result of Algorithm* B *and let* $a \cdot s$ *be any total state compatible with* $a \cdot \mathrm{t}^B$, *definite stable, and loop unstable. Assume that zero or more, but not all, indefinite gate vertices are marked. Assume also that every wire vertex between a marked and an unmarked indefinite gate vertex is unstable. Then there exists at least one unmarked indefinite gate vertex* k, *such that all indefinite wire vertices in* α^k *are unstable.*

Proof: Consider the directed graph $G = (\mathcal{V}', \mathcal{E}')$, where

$$\mathcal{V}' = \{i \in \mathcal{S} | \mathrm{t}_i^B = \Phi\}, \quad \text{and}$$
$$\mathcal{E}' = \{(i, j) \in \mathcal{E} | i \in \mathcal{G} \text{ or } s_i = S_i\,(a \cdot s)\}.$$

G can be obtained from the network graph by retaining only the indefinite vertices and those edges between indefinite vertices that are in the fanout set of vertices that are stable in $a \cdot s$. G has two important properties:

i. there is no path from a marked vertex to an unmarked vertex, and

ii. there is no cycle in G.

Both properties follow from the construction of G and the assumptions in the lemma.

Now consider a reverse path in G. Start at some unmarked gate vertex $k \in \mathcal{V}'$ and traverse G backwards. From (ii) and the fact that G is finite, it follows that a reverse path in G started at vertex k must stop at some vertex, say j. Note that j must be a gate vertex, and, by (i), must be unmarked. Furthermore, since each indefinite gate vertex has at least one indefinite wire vertex in its fan-in set, it follows that all indefinite wire vertices in α^j must be unstable; otherwise the reverse path would not have stopped at j. □

LEMMA 7: *Let* t^B *be the result of Algorithm* B *and let* $a \cdot s$ *be any total state compatible with* $a \cdot \mathrm{t}^B$, *definite stable, and loop unstable. If, for some indefinite gate vertex* j, *all indefinite vertices in* α^j *are unstable, then there exists a state* $\dot{s}$ *reachable from* s, *compatible with* t^B, *definite stable and loop unstable, such that*

i. $\dot{s}_j = \overline{s_j}$, *and*

ii. all indefinite wire vertices in β^j *are unstable in* $\dot{s}$.

Proof: We construct $\dot{s}$ in two steps. First we show that there is a state $\tilde{s}$ reachable from s such that $\tilde{s}_j = \overline{S_j\,(a \cdot \tilde{s})}$. We then show how to reach $\dot{s}$ from $\tilde{s}$.

For every input delay and gate vertex k, let $\tilde{s}_k = s_k$. If $k \neq j$, let $\alpha^k\,(\tilde{s}) = \alpha^k\,(s)$. Let $\alpha^j\,(\tilde{s}) = c^j$, where $c^j \in \{0, 1\}^{d_j}$ is such that

$c^j \sqsubseteq \alpha^j(\mathrm{t}^B)$ and $S_j(a \cdot c) = \overline{b_j}$ for all $c \in \{0, 1\}^m$ such that $\alpha^j(c) = c^j$. We claim that such c^j is guaranteed to exist. Suppose it did not, *i.e.*, assume that, for all $c^j \in \{0, 1\}^{d_j}$ such that $c^j \sqsubseteq \alpha^j(\mathrm{t}^B)$, there is some state $w \in \{0, 1\}^m$ such that $c^j = \alpha^j(w)$ and $S_j(a \cdot w) = w_j$. Since S_j depends only on the vertices in α^j, we can conclude that $S_j(a \cdot w) = w_j$ implies $S_j(a \cdot w') = w_j$ for every $w' \in \{0, 1\}^m$ such that $\alpha^j(w) = \alpha^j(w')$. Altogether, we have that $S_j(a \cdot w) = w_j$ for every $w \in \{0, 1\}^m$ such that $\alpha^j(w) \sqsubseteq \alpha^j(\mathrm{t}^B)$. By the definition of ternary extension, this implies that $\mathrm{S}_j(a \cdot \mathrm{t}^B) = w_j$. But, by Lemma 3, $\mathrm{S}(a \cdot \mathrm{t}^B) = \mathrm{t}^B$; thus $\mathrm{t}^B_j = w_j \in \{0, 1\}$. This contradicts the assumption that j is an indefinite gate vertex. Hence, our claim that such c^j exists is true.

It remains to be shown that $sR^*_a\,\tilde{s}$. However, this follows from the fact that $c^j \sqsubseteq \alpha^j(\mathrm{t}^B)$, the fact that all indefinite vertices in α^j are unstable, and the fact that s is compatible with t^B.

We now are ready to construct $\dot{s}$. For every input and gate vertex i let

$$\dot{s}_i = \begin{cases} S_j(a \cdot \tilde{s}) & \text{if} \quad i = j, \\ \tilde{s}_i & \text{otherwise,} \end{cases}$$

and

$$\beta_i(\dot{s}) = \begin{cases} \beta^j(S(a \cdot \tilde{s})) & \text{if} \quad i = j, \\ \beta^i(\tilde{s}) & \text{otherwise.} \end{cases}$$

Clearly, $\tilde{s}R^*_a\,\dot{s}$ and thus $sR^*_a\,\dot{s}$. By construction, $\tilde{s}_j = S_j(a \cdot \tilde{s}) = \overline{\tilde{s}_j}$, and Property (i) holds. On the other hand, the construction of $\tilde{s}$ ensures that every wire vertex in the fan-out set of gate j will be unstable, since we simultaneously set their outputs to their current excitations and change their inputs. Thus Property (ii) follows. If gate vertex j is indefinite, then each wire vertex in β^j is also indefinite. Consequently, it is straightforward to verify that $\tilde{s}$ is definite stable, loop unstable, and compatible with t^B. □

LEMMA 8: *Let t^B be the result of Algorithm* B *and let $a \cdot s$ be any total state compatible with $a \cdot \mathrm{t}^B$, definite stable, and loop unstable. Assume there are r indefinite gate vertices. Then, for each k, $0 \leq k \leq r$, there is a state $s^k \in \{0, 1\}^m$ with k vertices marked such that $sR^*_a\,s^k$ and $a \cdot s^k$ is compatible with $a \cdot \mathrm{t}^B$, definite stable, loop unstable, and every wire vertex between a marked and an unmarked indefinite gate vertex is unstable.*

Proof: We proceed by induction on the number of indefinite gate vertices which have been marked, *i.e.*, complemented. For the basis, $k = 0$, let $s^0 = s$ and the claim follows trivially. Now assume inductively that the claim holds for $k \geq 0$. By Lemma 6, it follows that there exists an unmarked indefinite gate vertex j such that all indefinite gate vertices in α^j are unstable in $a \cdot s^k$. But Lemma 7 guarantees the existence of a state s^{k+1}, such that $s^k R_a^* s^{k+1}$ and $a \cdot s^{k+1}$ is compatible with $a \cdot \mathrm{t}^B$, definite stable, and loop unstable. Furthermore, gate vertex j is complemented, and all the indefinite wire vertices in β^j are unstable in $a \cdot s^{k+1}$. Now mark vertex j, and note that all indefinite wire vertices between marked and unmarked indefinite gate vertices are still unstable. Hence, the induction step goes through and the lemma follows. □

COROLLARY 4: *Let t^B be the result of Algorithm B and let $a \cdot s$ be any total state compatible with $a \cdot \mathrm{t}^B$, definite stable, and loop unstable. Then there is a state $\tilde{s}$ reachable from s and such that $a \cdot \tilde{s}$ is compatible with $a \cdot \mathrm{t}^B$, definite stable, loop unstable, and all indefinite gate vertices have complementary values in s and $\tilde{s}$.*

Proof: This follows immediately from Lemma 8 for k equal to the number of indefinite gate vertices. □

We are now ready to state and prove the main result of this section:

LEMMA 9: *Let N and* **N** *be as in Theorem 2. Then there exists a non-transient cycle Z which is reachable from the initial state b such that*

$$lub\,\{s | s \in Z\} \sqsupseteq \mathrm{t}^B.$$

Proof: By Lemmas 3 and 5, and Corollary 3, it follows that $a \cdot t^B$ is compatible with $a \cdot \mathrm{t}^B$, definite stable, and loop unstable. Hence, Corollary 4 can be applied. Since Corollary 4 can be applied any number of times and there is only a finite number of possible states, there must exist a cycle in the R_a graph. By the construction of Corollary 4 it follows that each indefinite gate vertex oscillates. By the construction in Lemmas 7 and 8, it is also easy to see that every indefinite wire vertex in the fanout sets of the indefinite gate vertices also oscillates. However, a wire vertex j in the fan-out set β^i of some gate vertex i is indefinite if and only if gate vertex j is indefinite. Hence, every indefinite vertex is oscillating. Since all definite vertices are stable, it follows that the cycle is non-transient. □

REFERENCES

1. J. H. Anderson and M. G. Gouda, A new explanation of the glitch phenomenon, *Acta Informatica*, 1991, *28*, pp. 297-309.
2. J. A. Brzozowski and J. C. Ebergen, Recent developments in the design of asynchronous circuits. In J. Demetrovics J. Csirik and F. Gécseg, editors, *Proceedings of Fundamentals of Computation Theory*, Lecture Notes in Computer Science, Berlin, Germany, August 1989. Springer-Verlag, pp. 78-94.
3. J. A. Brzozowski and J. C. Ebergen, On the delay-sensitivity of gate networks, *IEEE Transactions on Computers*, November 1992, *41*, pp. 1349-1360.
4. J. A. Brzozowski and C.-J. H. Seger, A characterization of ternary simulation of gate networks, *IEEE Transactions on Computers*, November 1987, *C-36, 11*, pp. 1318-1327.
5. J. A. Brzozowski and C.-J. H. Seger, Advances in asynchronous circuit theory part I: Gate and unbounded inertial delay models, *Bulletin of the European Association of Theoretical Computer Science*, 1990, *42*, pp. 198-249.
6. J. A. Brzozowski and C.-J. H. Seger, Advances in asynchronous circuit theory part II: Bounded inertial delay models, MOS circuits, design techniques, *Bulletin of the European Association of Theoretical Computer Science*, 1991, *43*, pp. 199-263.
7. J. A. Brzozowski and C.-J. H. Seger, *Asynchronous Circuits*, Springer-Verlag, to appear.
8. J. A. Brzozowski and M. Yoeli, *Digital Networks*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1976.
9. J. A. Brzozowski and M. Yoeli, On a ternary model of gate networks, *IEEE Transactions on Computers*, 1979, *C-28, 3*, pp. 178-184.
10. E. B. Eichelberger, Hazard detection in combinational and sequential switching circuits, *IBM Journal of Research and Development*, 1965, *9*, pp. 90-99.
11. D. A. Huffman, The synthesis of sequential switching circuits, *IRE Transactions on Electronic Computers*, 1954, *257, 3*, pp. 161-190.
12. D. A. Huffman, The synthesis of sequential switching circuits, *IRE Transactions on Electronic Computers*, 1954, *257, 4*, pp. 275-303.
13. Z. Kohavi, *Switching and Finite Automata Theory, Second Edition*, McGraw-Hill Book Company, New York, New York, USA, 1978.
14. M. M. Mano, *Digital Design, Second Edition*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1991.
15. E. J. McCluskey, Fundamental mode and pulse mode sequential circuits. In C. M. Popplewell, editor, *Proceedings of the IFIP Congress 62*, Amsterdam, The Netherlands, 1963, IFIP, North-Holland Publishing Company, pp. 725-730.
16. E. J. McCluskey, *Logic Design Principles*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1986.
17. C. E. Molnar, T. P. Fang and F. U. Rosenberger, Synthesis of delay-insensitive modules, In H. Fuchs, editor, *Proceedings of the 1985 Chapel Hill Conference on VLSI*, Rockville, Maryland, USA, 1985. Computer Science Press, pp. 67-86.
18. M. Mukaidono, Regular ternary logic functions – ternary logic functions suitable for treating ambiguity, In *Proceedings of the 13th Annual Symposium on Multiple-Valued Logic*, Los Angeles, California, USA, May 1983. IEEE, Computer Society Press, pp. 286-291.
19. D. E. Muller and W. S. Bartky, A theory of asynchronous circuits, In *Proceedings of an International Symposium on the Theory of Switching*, Annals of the Computation Laboratory of Harvard University, Cambridge, Massachusetts, USA, 1959. Harvard University, Harvard University Press, pp. 204-243.

20. C. E. SHANNON, A symbolic analysis of relay and switching circuits, *AIEE Trans.*, 1938, *57*, pp. 713-723.
21. S. H. UNGER, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, New York, New York, USA, 1969.
22. M. YOELI and I. REICHER, *Synthesis of delay-insensitive circuits based on marked graphs*, Technical Report 543, Department of Computer Science, Technion, Haifa, Israel, 1989.
23. M. YOELI and S. RINON, Application of ternary algebra to the study of static hazards, *Journal of the ACM*, 1964, *11*, *1*, pp. 84-97.