On the delay-sensitivity of gate networks

On the Delay-Sensitivity of Gate Networks

by

J.A. Brzozowski J.E. Ebergen

90/5

July, 1990

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

On the Delay-Sensitivity of Gate Networks*

J.A. Brzozowski†
Department of Mathematics and Computing Science
Eindhoven University of Technology
Eindhoven, The Netherlands

J.C. Ebergen
Computer Science Department
University of Waterloo
Waterloo, Ontario, Canada

Abstract

In classical switching theory it is usually assumed that asynchronous sequential circuits are operated in the *fundamental mode*. In this mode, a circuit is started in a stable state; then the inputs are changed to cause a transition to another stable state. The inputs are not allowed to change again until the entire circuit has stabilized. In contrast to this, delay-insensitive circuits —the correctness of which is insensitive to delays in their components and wires— use the *input-output mode*. Here, it is assumed that inputs may change again, in response to an output change, even before the entire circuit has stabilized. In this paper, we show that such commonly used behaviors as those of the set-reset latch and Muller's C-ELEMENT do not have delay-insensitive realizations, if gates are used as the basic components. In fact, we prove that no nontrivial sequential behavior with one binary input possesses a delay-insensitive realization using gates only. Our proof makes use of the equivalence between ternary simulation and the General Multiple Winner model of circuit behavior.

1 Introduction

Asynchronous circuits have witnessed a remarkable revival during the past decade. This revival came about on two fronts. On the one hand, important results were found concerning the analysis of asynchronous circuits using the 'classical' approach [3, 4, 18]. On the other hand, new formal approaches have been developed and applied in the design of special types of asynchronous circuits, such as speed-independent circuits [7, 16], delay-insensitive circuits [2, 8, 15], quasi delay-insensitive circuits [11], and self-timed systems [6, 19]. Successful applications

*This work was supported by the Dutch organization for scientific research, NWO, and by the Natural Sciences and Engineering Research Council of Canada under Grants A0871 and OGP0041920.

†On leave during 1989-1990 from the Computer Science Department, University of Waterloo, Waterloo, Ontario, Canada.

of these new approaches have been demonstrated by A. J. Martin [10], C. van Berkel et al. [21] and, most recently, by I. E. Sutherland in his Turing Award lecture [20].

There are some important differences between the 'classical' and the 'new' approaches. One of these differences lies in the 'mode of operation' of a network which prescribes how the environment should interact with a network in order to obtain the desired input-output behavior. In the classical approach one applies the so-called fundamental mode operation [13] in which the environment changes the inputs and holds them fixed until the network has stabilized completely. Only after the network has reached a stable state, is the environment allowed to give the next input change. In the new approaches, one applies the so-called input-output mode of operation [2, 15] in which the environment does not have to wait until the network has stabilized completely to give the next input change: an input change may be made by the environment as soon as the network has given an appropriate response to a previous input change.

Another difference between the classical and the new approaches lies in the different basic formalism. In classical asynchronous circuit design, the basic formalism is Boolean algebra, where Boolean functions are the essential objects. In the new approaches, one applies an event-based formalism, where sequences of events are the essential objects. Examples of event-based formalisms are trace theory [7, 8, 17] and Petri nets [6, 14, 22]. The use of different formalisms leads also to the use of different sets of primitive elements. In the classical approach, the primitive elements are the logic gates, which correspond to the primitive Boolean functions. In the new approaches, the primitive elements are such elements as the Muller C-ELEMENT, TOGGLE, MERGE, and ARBITER, which correspond to basic sequences of events.

The input-output mode of operation and primitive elements, like C-ELEMENTs and TOGGLEs, are used in delay-insensitive circuits. Informally, a delay-insensitive circuit is one whose correctness is insensitive to delays in the wires and primitive elements. A fundamental question is whether a different set of primitive elements for the design of these circuits is really necessary, or can every delay-insensitive circuit be realized by a network using gates only? For example, does the C-ELEMENT or TOGGLE have a delay-insensitive gate realization? Although it is generally assumed that such realizations do not exist, no proof of this has been given so far. We provide such a proof in this paper. In fact, we prove a more general result that no nontrivial sequential behavior with one binary input has a delay-insensitive gate realization. In doing so, we explain the differences between the fundamental mode and the input-output mode using one formal framework that combines the event-based and state-based approach.

2 Specification of Input-Output Behavior

When specifying the behavior of a network to be designed, we need to describe how the proposed network is to communicate with its environment through its input and output terminals. In this paper, we formally specify an *input-output behavior* by a 5-tuple $B = \langle h, k, S, T, s_0 \rangle$, where

- $h \geq 0$, is the *number of input variables*,
- $k \geq 0$, is the *number of output variables*,

- $S \subseteq \{0,1\}^{h+k}$, is the *state set*,
- $T \subseteq (S \times S) - \{(s,s) \mid s \in S\}$, is the *transition set*, and
- $s_0 \in S$ is the *initial state*.

The vector $u = u_1, \ldots, u_h$ will be used to represent the input variables, and $v = v_1, \ldots, v_k$ will denote the output variables. The vector $u_1, \ldots, u_h, v_1, \ldots, v_k$ represents the *state* of B. A state $c \in \{0,1\}^{h+k}$ is sometimes written $c = ab$, where $a \in \{0,1\}^h$ and $b \in \{0,1\}^k$; this permits us to identify the input and output components of c. A transition $(ab, a'b')$ is said to be an *input transition*, if only the inputs have changed, i.e., if $a \neq a'$ and $b = b'$; it is an *output transition*, if only the outputs have changed, i.e., if $a = a'$ and $b \neq b'$. Note that a transition may be neither an input transition nor an output transition, namely, when $a \neq a'$ and $b \neq b'$. A state of the input-output behavior is said to be *stable*, if it has only input transitions leaving it or has no transitions at all; otherwise, it is *unstable*.

To illustrate the definitions given above, we present a number of examples.

Example 1. An input-output behavior of the set-reset latch, under the assumption that the set and reset inputs are never 1 at the same time, is given in Figure 1. The latch has binary inputs u_1 ('set') and u_2 ('reset') and a binary output v. The state set consists of all binary triples $u_1 u_2 v$, except 110 and 111, and the transitions are shown in Figure 1. For convenience, we label each transition with the variable that changes during the transition. The initial state is 000. (For brevity, we often write tuples without commas; for example, 110 represents (1,1,0).) □

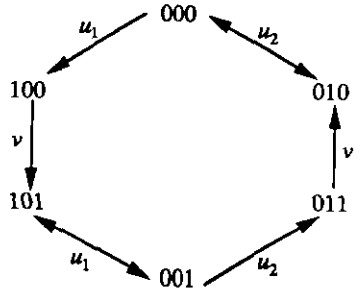

Figure 1: Input-output behavior of latch with $u_1 u_2 = 11$ disallowed.

Example 2. The graph of Figure 2 shows an input-output behavior of the latch where $u_1 u_2 = 11$ is permitted, but where u_1 and u_2 never change simultaneously. Notice that Figure 1 is symmetrical, but Figure 2 is not. This asymmetry is introduced in the states where $u_1 u_2 = 11$ by giving the 'reset' priority over the 'set.' □

Example 3. In Figure 3 we show an input-output behavior of the C-ELEMENT with inputs u_1 and u_2 and output v. □

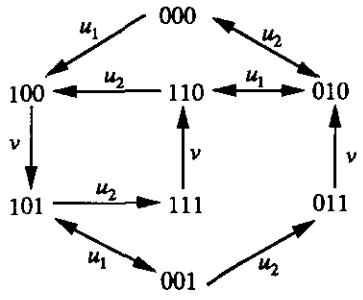

Figure 2: Behavior of latch with $u_1 u_2 = 11$ allowed.

Example 4. Figure 4 shows an input-output behavior of a TOGGLE with input u and outputs v_1 and v_2. If we count the input changes starting from the initial state 000, each odd input change causes a change in output v_1 and each even input change causes a change in output v_2. □

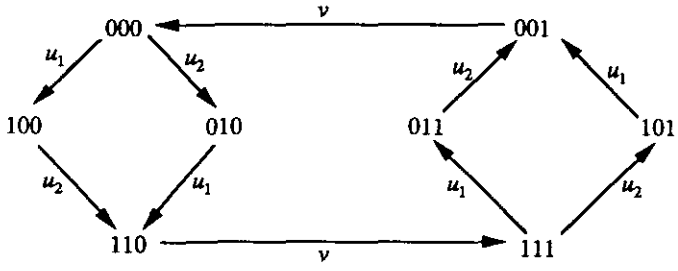

Figure 3: Input-output behavior of C-ELEMENT.

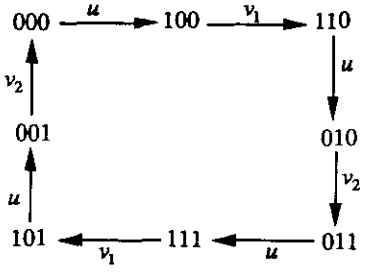

Figure 4: Input-output behavior of TOGGLE.

In this paper we study a restricted class of input-output behaviors. We consider only those input-output behaviors where (a) the initial state is stable; (b) each unstable state has exactly one outgoing transition, this is an output transition, and the state reached by this transition

is stable; (c) exactly one (input) variable changes in every input transition; and (d) exactly one (output) variable changes in every output transition. We call this class of input-output behaviors *simple deterministic*. Note that the behaviors in Figures 1, 2, 3, and 4 are simple deterministic.

The motivation for the assumptions above is the following. First, it would be unreasonable to start a network in an unstable state and expect it to operate properly; in fact, most practical designs provide for special 'reset' inputs to make sure that the network is properly initialized. Second, we are interested in deterministic behavior in many common circuits, such as counters, latches or C-ELEMENTS. Thus the final outcome of a transition should be a unique stable state. More will be said about this later, in connection with oscillations. We also point out that the first two assumptions are very much in line with the classical theory of fundamental-mode operation. The last two assumptions —that a single signal changes during any transition— are not restrictive, in view of the fact that we are interested in delay-insensitive networks. For such networks, one necessarily takes into account wire delays. In the presence of wire delays, it would be difficult to argue strongly in favor of a model where simultaneous changes play a significant role: a small deviation in a wire delay is sufficient to change a simultaneous occurrence of two signals into a sequential occurrence. Finally, the assumptions simplify a number of proofs.

3 The General Multiple Winner Model

In order to address the question whether a gate network realizes a certain input-output behavior, we first discuss the analysis of gate networks. We view a *gate network* [3] as a directed labeled graph $N = (n, m, V, E, g)$, where

- $n \geq 0$ is the *number of input variables*;
- $m \geq 0$ is the *number of gate variables* associated with the gates of N;
- $V = \{1, \ldots, n+m\}$ is the set of *vertices* of N corresponding to the n input terminals and m gates;
- $E \subseteq V \times V$, is the set of *edges* of N, corresponding to the wires connecting the inputs and gates;
- $g = g_1, \ldots, g_m$ is the vector of *Boolean functions* associated with the gates of N.

The vertices of N are classified as follows. The first n vertices, numbered $1, \ldots, n$, correspond to the input terminals and are all of indegree 0. Associate with input vertex i the variable x_i. The remaining vertices are all of indegree greater than 0; they correspond to the gates of the network and are numbered $n+1, \ldots, n+m$. Associate with gate vertex $n+i$ the variable y_i. Let the vectors $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_m$ denote the network inputs and the gate outputs, respectively. The vector xy represents the *state* of the network.

The function g_i is the *excitation* of gate i, mapping $\{0,1\}^{d_i}$ to $\{0,1\}$, where d_i is the indegree of vertex i. The variables on which g_i depends are shown by the edges in E that lead to vertex $n+i$. It is often convenient, however, to view g_i as a function of the entire state, i.e.,

to treat it as a mapping $g_i(xy)$ from $\{0,1\}^{n+m}$ to $\{0,1\}$. Notice that gates are not confined to simple AND and OR gates, but may implement *any* Boolean function.

In any given state ab the output b_i of gate i may differ from its excitation $g_i(ab)$. Gate i is then said to be *unstable*; otherwise it is *stable*. A state of the network is *stable* if all the gates are stable in that state.

To analyze a given network in any state we use the *General Multiple Winner* (GMW) model [5]. In this model the relation R_a defines the possible transitions among the network states on the basis of the unstable gates in each state. The relation R_a is defined for any (fixed) input vector $a \in \{0,1\}^n$. Let

$$D_a = \{ab \mid b \in \{0,1\}^m\},$$

i.e., let D_a represent all the total network states that have the input component fixed at $x = a$. For any d in D_a define the set of unstable gate indices to be

$$U(d) = \{i \mid d_{n+i} \neq g_i(d)\}.$$

Next, define R_a as the smallest relation on D_a with

$d\, R_a\, d$, if $U(d) = \emptyset$, i.e., if d is stable,
$d\, R_a\, d^W$, if W is a nonempty subset of $U(d)$,

where the notation d^W stands for the vector d in which all components with subscripts in W are complemented. Thus the relation R_a relates each stable state to itself and each unstable state to any state obtained by complementing a non-empty subset of the unstable gate variables. (A state with two or more unstable variables is said to have a *race*, and the gate variables with subscripts in W are the *multiple winners* of the race. The reader should also note that two notations are used in connection with relations, namely: aRb and $(a,b) \in R$.)

Example 5. To illustrate the GMW model, consider the network of the NOR latch and the corresponding graph in Figure 5. We have

$V = \{1,2,3,4\}$,
$E = \{(1,3),(2,4),(3,4),(4,3)\}$,
$x = x_1x_2$, $y = y_1y_2$,
$g_1(xy) = \overline{(x_1 + y_2)}$ and $g_2(xy) = \overline{(x_2 + y_1)}$.

The graph of the relation R_{00} is shown in Figure 6. Note that, after state 0000 is reached, the network may end up in stable state 0001 or in the stable state 0010 (this represents a *critical race*), or it may oscillate between the states 0000 and 0011. We consider such oscillations unacceptable and indicative of undesirable behaviors in a physical circuit. □

(a) (b)

Figure 5: NOR latch (a) Gate network and (b) graph.

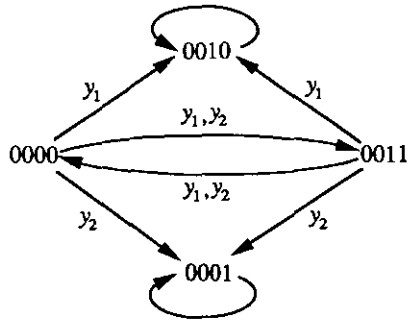

Figure 6: Relation R_{00}, partial network behavior of NOR latch started in state 0000.

In the GMW analysis of a network state, a cycle must be reached eventually in the graph of the relation R_a, since the number of network states is finite. Recall that every stable network state forms a cycle of length one. The transient states occurring during a transition from a given state are often of no interest. A cycle is *transient* if there exists a gate which is unstable in every state of the cycle and has the same value in all these states; otherwise it is *nontransient*. Let

$$cycl(R_a, d) = \{e \in D_a \mid dR_a{}^* e \text{ and } eR_a{}^+ e\}$$

be the set of all the states reachable from state d that appear in cycles of R_a. Here, $R_a{}^+$ and $R_a{}^*$ denote the transitive and the reflexive-and-transitive closures of R_a. Let $trans(R_a, d)$ be the set of all the states in $cycl(R_a, d)$ that appear only in transient cycles of the graph of R_a, and let

$$out(R_a, d) = cycl(R_a, d) - trans(R_a, d).$$

We interpret $out(R_a, d)$ as the final outcome of the GMW analysis of the network started in state d, where the inputs are kept fixed at a. The case of most interest is that in which the outcome consists of a single state. This state is necessarily stable and is reached from d in a finite time. Consequently, if the outcome consists of a single state, the network does not exhibit nontransient oscillatory behavior. As we stated before, nontransient oscillatory behaviors are indicative of undesirable behaviors in a physical circuit and are therefore unacceptable.

From the construction of the relation R_a it follows that an arbitrary finite delay is associated with each gate, but that wires have no delays. If we wish to represent wire delays, we do so by treating them as one-input, one-output gates performing the identity function. In fact, this is exactly what we do in Section 5, when we examine delay-insensitive realizations of input-output behaviors. There, a wire delay is included for each connection wire in a gate network.

4 Fundamental Mode Realization

In order to realize a specified input-output behavior, we need not only an appropriate network, but also a co-operating environment. Roughly speaking, the environment is expected to produce the 'correct' input changes at the 'correct' times. For example, consider the input-output behavior of the latch of Figure 1, started in state $u_1u_2v = 000$. When a 'set' input is applied to a network that should realize this input-output behavior, we expect the network to respond by changing output v and moving to a state whose input-output component is $u_1u_2v = 101$. If the 'set' input is then changed again, the network should move to input-output state 001, 'remembering' that a 'set'-pulse had been received. If the 'set' input changes too soon to 0, however, some networks may fail to 'remember' the 'set' and may change back to state 000. In the fundamental mode operation, the environment is allowed to change the 'set' input again only when the complete network has stabilized. The reader should note that the fundamental mode of operation requires that the environment must know somehow when the network is ready to receive the next input; however, it is irrelevant to the designer of a fundamental mode circuit how this environment restriction is implemented. In this section we make the concept of fundamental mode realization [13] precise.

Before giving a formal definition of fundamental mode realization, we need to define the restriction of a network state to a subset of its state variables. For a given input-output behavior B and a network N, we want to map each input of B to exactly one input of N and each output of B to exactly one gate variable of N. One could also view this as a 1-to-1 mapping of a subset of the input variables of N to the input variables of B and of a subset of the gate variables of N to the output variables of B. In case such a mapping exists, we say that there is a *restriction of N to B*. We extend this restriction to a mapping of the states of N to states of B by simply removing all state variables that are not representatives of state variables in B. The restriction of network state q to B is denoted by $q \downarrow B$. We also generalize this notion to the restriction of any binary relation R_N on the states of N to a binary relation $R_N \downarrow B$ on the states of B in the obvious way:

$$s\,(R_N \downarrow B)\,s' \text{ iff there exist states } q, q' \text{ of } N \text{ such that}$$

$$q \downarrow B = s,\ \ q' \downarrow B = s',\ \text{ and } qR_Nq'.$$

Given a restriction of a network N to a behavior B, we will simulate the network behavior step by step with the environment behavior as given by B, in order to check whether N is proper fundamental mode realization of B. First, if N is to realize B, there must exist an initial stable state q_0 of N whose restriction is s_0. Thus we will consider an initialized network (N, q_0) as a possible realization of B. After that, N should imitate B as follows. If, in B,

an input change is followed by an output change, then, in N, the corresponding input change *must* eventually result in the corresponding output change. Thus phenomena such as deadlock and livelock [9] in N are not tolerated. We will take appropriate precautions to make sure that our definition of realization does not permit such phenomena to occur. Furthermore, if an output of B is to change once, then the corresponding output of N should not change more that once; thus dynamic hazards are not acceptable. Similarly, if an output of B is *not* supposed to change, than neither is the corresponding output of N; consequently, static hazards are also excluded. We will show that our definition of realization is correct from this point of view. All the concepts mentioned above will be made precise in this section.

The details of the simulation are as follows. The set of network states that can be reached from q_0 is denoted by Q_F, and the possible transitions among these states are recorded in the relation R_F. Let $B = \langle h, k, S, T, s_0 \rangle$ be a simple deterministic behavior and let $N = \langle n, m, V, E, g \rangle$ be a network, and q_0 an initial state. The set $Q_F \subseteq \{0,1\}^{n+m}$ and binary relation $R_F \subseteq Q_F \times Q_F$ are defined inductively as follows:

Basis : $Q_F = \{q_0\}$, $R_F = \emptyset$.

Induction Step : For each $q \in Q_F$, where $s = q \downarrow B$ and $a \in \{0,1\}^n$ denotes the input vector of q, we have the following rules:

- *Rule 1*:
  If q is unstable, then, for each q' such that qR_aq',
  add q' to Q_F and (q, q') to R_F.
- *Rule 2*:
  If q is stable, then, for each input u_i of B such that $sTs^{\{i\}}$,
  add $q^{\{j\}}$ to Q_F and $(q, q^{\{j\}})$ to R_F,
  where $u_i = x_j$ in the restriction of N to B.

(Recall that $q^{\{j\}}$ denotes state q with gate variable j complemented.) The induction step is applied until Q_F and R_F can no longer be enlarged.

Note that, when B is simple deterministic and s is unstable, s has only one output transition. Consequently, there exists no input u_i of B such that $sTs^{\{i\}}$, and any stable state $q \in Q_F$ with $q \downarrow B = s$ will not be related to any state by R_F.

A tentative definition of '(N, q_0) is an F-realization of B' might be that

$$R_F \downarrow B - I = T$$

must hold, where I is the identity relation on S and '$\downarrow$' has a higher priority than '$-$'. (Notice that self-loops may be created when R_F is restricted to B. Since such self-loops are not allowed in T, we subtract them from the restriction.) This definition is not quite satisfactory, however, since phenomena similar to deadlock and livelock may occur in the realization. These phenomena are illustrated in the following examples.

Example 6. Let behavior B_1 have input u, output v, initial state 00, and S_1 and T_1 are defined by

$$B_1 : 00 \xrightarrow{u} 10 \xrightarrow{v} 11 \xrightarrow{u} 01 \xrightarrow{v} 00 \,,$$

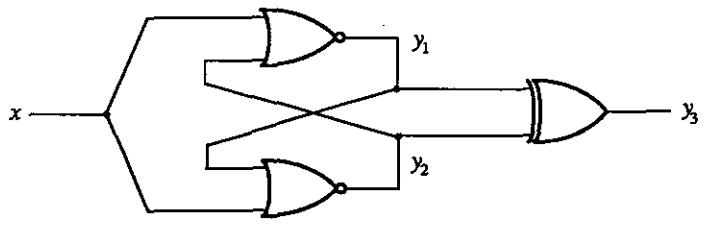

Figure 7: Network N_2.

where the last state is the equal to the initial state. Let network N_1 have input x and two gates g_1, g_2 with outputs y_1, y_2. The functions of the gates are given by

$$g_1(xy) = y_1 + y_2\overline{x} \text{ and } g_2(xy) = x\overline{y_1} + \overline{y_1}y_2.$$

Let the restriction be defined by $u = x$ and $v = y_2$. Furthermore, let $q_0 = 000$; this is a stable network state. Applying the above construction for Q_F and R_F, we find the following. Since $q_0 = 000$ is stable, Rule 2 can be applied. There is only one input transition (00,10) in B_1, which gives rise to (000,100) in R_F. State 100 is unstable, and Rule 1 yields (100,101) in R_F. State 101 is again stable, and Rule 2 with input transition (11,01) from B_1 gives (101,001) in R_F. In state 001, y_1 is unstable, so Rule 1 gives (001,011) in R_F. State 011 is also unstable, and Rule 1 yields (011,010) in R_F. Next, state 010 is stable, and Rule 2 now gives (010,110) in R_F. The last state 110 is also stable, and the simulation ends here. In summary, we have

$$000 \xrightarrow{u} 100 \xrightarrow{v} 101 \xrightarrow{u} 001 \xrightarrow{y_1} 011 \xrightarrow{v} 010 \xrightarrow{u} 110.$$

Although the condition $R_F \downarrow B_1 - I = T_1$ is satisfied, a phenomenon similar to deadlock occurs: as soon as state 110 is reached, no new output can ever be produced. □

The previous example illustrated that, although a network may stabilize, there is no guarantee that it will produce the required output. The following example illustrates that a network may even fail to stabilize.

Example 7. Consider the behavior B_2 with input u, output v, $s_0 = 10$, and S_2 and T_2 given by

$$B_2 : 10 \xrightarrow{u} 00 \xrightarrow{v} 01.$$

The network of Figure 7 shows the NOR latch of Figure 5, but now with the inputs tied together and the outputs of the two NOR gates connected to a XOR gate. The restriction of this network N_2 to B_2 is given by $u = x$ and $v = y_3$. The initial state of N_2 is 1000. We find R_F as shown in Figure 8, and $R_F \downarrow B_2 = \{(10,00),(00,00),(00,01)\}$. Consequently, $R_F \downarrow B_2 - I = T_2$. We also observe, however, that the network may keep oscillating between states 0000 and 0110 and thus never reach state 0011 or 0101. In other words, if the network fails to stabilize, the required output $v = 1$ may never be produced. This phenomenon is similar to livelock. □

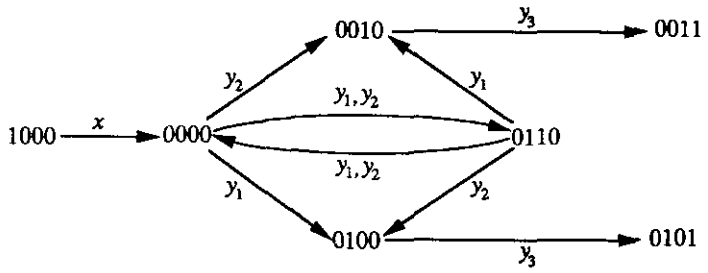

Figure 8: Relation R_F for N_2.

In order to avoid a realization like that in Example 6, we introduce the notion of F-consistency. We say that (N, q_0) with restriction $\downarrow$ is *F-consistent with respect to* B iff, for all $q \in Q_F$,

$$q \downarrow B \text{ is unstable } \Rightarrow q \text{ is unstable.}$$

The F-consistency condition requires that, as long as an output change is expected with respect to B, network N will remain unstable. Notice that, in Example 6 above, 100 and 110 both map onto the unstable state 10; state 100 is unstable, but 110 is not. Consequently, $(N_1, 000)$ is not F-consistent with respect to B_1.

In order to guarantee stabilization, we require the absence of nontransient oscillations. For this purpose we introduce the notion of F-determinism as follows. We say that (N, q_0) with restriction $\downarrow$ is *F-deterministic with respect to* B iff for all $q \in Q_F$, the result $out(R_a, q)$ of the GMW analysis consists of only one state, where a denotes the input vector of q. The F-determinism condition requires that the GMW analysis of any state in Q_F is guaranteed to stabilize in exactly one state. Obviously, if there exists a nontransient oscillation in a GMW analysis of a state q in Q_F, then $out(R_a, q)$ contains more than one state. Consequently, if the F-determinism condition is satisfied, then there are no nontransient oscillations. Notice that, in Example 7 above, $out(R_0, 0000) = \{0011, 0101, 0000, 0110\}$. Consequently, $(N_2, 1000)$ is not F-deterministic with respect to B_2.

It was proved in [3] that, if a delay is included in the circuit model in each connection wire (i.e., if the gate-and-wire-delay model is used), then there exists a nontransient oscillation iff $out(R_a, q)$ consists of more than one state. Consequently, for the gate-and-wire delay model, the absence of nontransient oscillations is equivalent to F-determinism.

The definition of F-realization now reads as follows:

Definition 1 *An initialized network (N, q_0) with restriction $\downarrow$ is an F-realization of behavior B iff (N, q_0) is F-consistent and F-deterministic with respect to B, and R_F satisfies the condition:*

$$R_F \downarrow B - I = T,$$

where I is the identity relation on S. □

We illustrate this definition by additional examples.

Example 8. Consider the behavior B_3 with input u, output v, initial state $s_0 = 00$, and S_3

and T_3 given in Figure 9. For the network N_3, we consider the NOR latch of Figure 5 again, now started in state 0010. Let the restriction of N_3 to B_3 be defined by $u = x_1$ and $v = y_2$.

$$B_3 : 00 \xrightarrow{u} 10 \xrightarrow{v} 11 \xrightarrow{u} 01$$

Figure 9: Input-output behavior B_3.

Performing the construction for Q_F and R_F, we obtain

$$\begin{aligned} R_F &= \{(0010, 1010), (1010, 1000), (1000, 1001), (1001, 0001)\} \\ R_F \downarrow B_3 &= \{(00, 10), (10, 10), (10, 11), (11, 01)\}. \end{aligned}$$

One verifies that every state $q \in Q_F$, for which $q \downarrow B_3$ is unstable in B_3, q is also unstable in N_3. Consequently, $(N_3, 0010)$ is F-consistent with respect to B_3. Furthermore, $(N_3, 0010)$ is F-deterministic with respect to B_3, and $R_F \downarrow B_3 - I = T_3$, i.e., $(N_3, 0010)$ is an F-realization of B_3. □

Example 9. As a second example, consider behavior B_3 with the network N_4 of Figure 10. The network has input x, gate variables y_1, y_2, y_3, and excitation functions $g_1 = x$, $g_2 = y_3$ and $g_3 = y_1 + y_2$. Assuming the network starts in state 0000, and that the restriction is defined by $u = x$ and $v = y_3$, we obtain

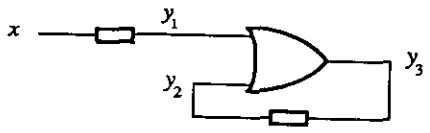

Figure 10: Network N_4.

$$R_F = \{(0000, 1000), (1000, 1100), (1100, 1101), (1101, 1111), (1111, 0111), (0111, 0011)\}.$$

Again, one verifies that $(N_4, 0000)$ is F-consistent and F-deterministic with respect to B_3, and $R_F \downarrow B_3 - I = T_3$, i.e., N_4 is an F-realization of B_3. □

For simple deterministic behaviors, F-realization satisfies Property 2 below with respect to static and dynamic hazards. A *static hazard* is said to occur on an output of a network during a sequence of state transitions in the relation R_a, if that output has the same value in the initial and final states of the sequence, but the complementary value in at least one intermediate state. Here the output is not supposed to change but does change (at least twice). A *dynamic hazard* is defined similarly, except that now the output has different values in the initial and final states. Here the output is supposed to change once, but changes more than once. Before formulating the property, we introduce some terminology. A sequence $q_0, .., q_k$, $k \geq 0$, of network states with input component fixed at a is an R_a-*sequence* if $q_i \, R_a \, q_{i+1}$ for $i = 0, .., k-1$. The restriction of an R_a-sequence to B will be called an $(R_a \downarrow B)$-*sequence*.

Property 2 *Let network (N, q_0) with restriction $\downarrow$ be an F-realization of simple deterministic behavior B, and let s be any state of B.*

- *s is stable iff for all states $q \in Q_F$, such that $q \downarrow B = s$, and for any R_a-sequence from q to $q' \in out(R_a, q)$, there is no change in the corresponding $(R_a \downarrow B)$- sequence (i.e., there are no static hazards).*
- *s is unstable iff for all states $q \in Q_F$ such that $q \downarrow B = s$, and for any R_a-sequence from q to $q' \in out(R_a, q)$, there is exactly one change in the corresponding $(R_a \downarrow B)$-sequence (i.e., the sequence has the form $s, \ldots, s, s', \ldots, s'$, where $s' = q' \downarrow B$, and there are no dynamic hazards).*

Proof. Let (N, q_0) be an F-realization of B and let s be any state of B.

For the first claim, because $R_F \downarrow B - I = T$, the condition

> for all $q \in Q_F$, with $q \downarrow B = s$, and any R_a-sequence starting in q, the corresponding $(R_a \downarrow B)$-sequence has no changes

is equivalent to the condition

> state s has no output transitions.

By the definition of stability and the fact that B is simple deterministic, this is equivalent to the condition that s is stable.

For the second claim we observe the following. Assume that for all $q \in Q_F$, with $q \downarrow B = s$, and any R_a-sequence starting in q, the corresponding $(R_a \downarrow B)$-sequence has exactly one change. Then there is an output transition from s in $R_F \downarrow B$. Thus, by the definition of unstable state, s is unstable.

Conversely, assume that s is unstable. Then s has an output transition in T and, since $R_F \downarrow B - I = T$, also in $R_F \downarrow B$. Let $q \in Q_F$, with $q \downarrow B = s$, and take an arbitrary R_a-sequence that starts in q and ends in a stable network state. Such a stable network state exists, since (N, q_0) is F-deterministic with respect to B. Since (N, q_0) is also F-consistent with respect to B and s is unstable in B, the final stable network state of the R_a-sequence does not map onto s. Hence the $(R_a \downarrow B)$-sequence has at least one change.

Suppose further that the first state different from s in the $(R_a \downarrow B)$-sequence is s'. Since B is simple deterministic and, therefore, s' is a stable state, all successors of s' in the $(R_F \downarrow B)$-sequence are also s'. Consequently, there is exactly one change in any $(R_a \downarrow B)$-sequence. Thus, the second claim also holds. □

5 Input-Output Mode Realization

For the design of *delay-insensitive circuits* [15], one also starts with a specification of an input-output behavior, but now the network is operated in *input-output mode*. This mode of operation stipulates that the environment is allowed to change the inputs of the network before the complete network has stabilized, as long as this is done in accordance with the state transitions of the input-output behavior; the network must produce the outputs as specified

in the input-output behavior. For example, for the input-output behavior of the TOGGLE of Figure 4, the environment can start by changing the input u. As soon as the network for the TOGGLE responds with a change in output v_1, the environment is allowed to change the input u again, etc. The reader should observe that the input-output mode places less responsibility on the environment, than does the fundamental mode: the environment can supply the next input change as soon as the network has produced an output change, and no additional information is needed.

When a network N is operated in input-output mode with respect to a behavior B, we construct the set of states Q_{IO} and the relation R_{IO}. The construction of these sets is similar to the construction of Q_F and R_F when a network N is operated in fundamental mode with respect to a behavior B. Again we assume that there exists a restriction $\downarrow$ and a stable state q_0 of N that maps onto the initial state s_0 of B. The set Q_{IO} and relation R_{IO} are defined inductively as follows:

Basis : $Q_{IO} = \{q_0\}$, $R_{IO} = \emptyset$.

Induction Step : For each $q \in Q_{IO}$, where $s = q \downarrow B$ and $a \in \{0,1\}^n$ denotes the input vector of q, we have the following rules:

- *Rule* 1':
  If $q \downarrow B$ is unstable, then, for each q' such that qR_aq',
  add q' to Q_{IO} and (q, q') to R_{IO}.
- *Rule* 2':
  If $q \downarrow B$ is stable, then for each input u_i of B such that $sTs^{\{i\}}$
  and for each nonempty subset W of $(U(q) \cup \{j\})$,
  where $u_i = x_j$ in the restriction of N to B,
  add q^W to Q_{IO} and (q, q^W) to R_{IO}.

The induction step is applied until Q_{IO} and R_{IO} can no longer be enlarged.

Notice that, for constructing Q_{IO} and R_{IO}, we examine the stability of $q \downarrow B$, instead of q.

The definition of IO-realization now reads as follows:

Definition 3 *An initialized network (N, q_0) with restriction $\downarrow$ is an IO-realization of behavior B iff (N, q_0) is IO-consistent, IO-deterministic, and R_{IO} satisfies the condition:*

$$R_{IO} \downarrow B - I = T,$$

where I is the identity relation on S and IO-consistency and IO-determinism are defined like F-consistency and F-determinism, but now with respect to Q_{IO}. □

The conditions that (N, q_0) be IO-consistent and IO-deterministic are introduced for the same reasons as in the case of F-realization.

Example 10. We can apply the definition of IO-realization to the behaviors and networks given in Examples 6, 7, and 8. The construction of Q_{IO} and R_{IO} happens to be exactly the same as for Q_F and R_F, in these three cases. We obtain the following conclusions:

- $(N_1, 000)$ is not an IO-realization of B_1, since $(N_1, 000)$ is not IO-consistent.
- $(N_2, 1000)$ is not an IO-realization of B_2, since $(N_2, 1000)$ is not IO-deterministic.
- $(N_3, 0010)$ is an IO-realization of B_3, since $(N_3, 0010)$ is IO-consistent and IO-deterministic and $R_{IO} \downarrow B_3 - I = T_3$.

□

Example 11. The situation is different for B_3 and N_4 from Example 9. Although $(N_4, 0000)$ is an F-realization of B_3, it is not an IO-realization of B_3. The following sequence of transitions can occur in R_{IO}:

$$0000 \xrightarrow{u} 1000 \xrightarrow{y_1} 1100 \xrightarrow{v} 1101 \xrightarrow{u} 0101 \xrightarrow{y_1} 0001 \xrightarrow{v} 0000.$$

From this sequence it follows that $R_{IO} \downarrow B_3 - I \neq T_3$, since $(01, 00) \notin T_3$. □

The property with respect to static and dynamic hazards also holds for IO-realizations.

Property 4 *Property 2 also holds when (N, q_0) with restriction $\downarrow$ is an IO-realization of simple deterministic behavior B.*

Proof: The proof of Property 2 still holds if each occurrence of F is replaced by IO. □

We prove below that, if a network IO-realizes a behavior, then it also F-realizes it. Consequently, the fundamental mode of operation can be seen as a special case of the input-output mode of operation. Thus, if an input-output behavior is not realizable by any network operated in fundamental mode, then certainly it is not realizable by any network operated in input-output mode.

Theorem 5 *If (N, q_0) with restriction $\downarrow$ IO-realizes B, then (N, q_0) with the same restriction also F-realizes B.*

Proof: Let (N, q_0) be an IO-realization of B. First, we prove that $R_F \subseteq R_{IO}$ and then that $(R_{IO} \downarrow B - I) \subseteq (R_F \downarrow B - I)$. From these two properties it follows that $(R_{IO} \downarrow B - I) = (R_F \downarrow B - I)$. Furthermore, if $R_F \subseteq R_{IO}$ and (N, q_0) is IO-consistent and IO-deterministic, it follows that (N, q_0) is F-consistent and F-deterministic as well. Consequently, we conclude that (N, q_0) is also an F-realization of B.

We prove $R_F \subseteq R_{IO}$ by induction on the construction of R_F and R_{IO}.
Basis: $\{q_0\} = R_F$ and $\{q_0\} = R_{IO}$.
Induction Step: R_F can be enlarged by applying Rule 1 or 2 to a state $q \in Q_F$. R_{IO} can be enlarged by applying Rule 1′ or 2′ to state q. Rule 1 applies when q is unstable. Suppose we can add (q, q') to R_F, because qR_aq'. Then we can also add (q, q') to R_{IO} by using Rule 1′, if $q \downarrow B$ is unstable, or by using Rule 2′, if $q \downarrow B$ is stable. Therefore, if R_F can be enlarged by Rule 1, then R_{IO} can be enlarged similarly.

If Rule 2 is used, then q is stable. Because (N, q_0) is IO-consistent, and q is in R_{IO} by the induction hypothesis, $q \downarrow B$ is also stable. Hence Rule 2′ applies, and R_{IO} can be enlarged like R_F by choosing W to be $\{j\}$. This completes the proof of the first claim, i.e., we have

$$Q_F \subseteq Q_{IO} \text{ and } R_F \subseteq R_{IO}.$$

The proof of the second claim, i.e., that $R_{IO} \downarrow B - I \subseteq R_F \downarrow B - I$, is also done by induction on the construction of R_{IO} and R_F.
Basis: Obviously, the second claim holds for $Q_{IO} = \{q_0\}$, $R_{IO} = \emptyset$ and $Q_F = \{q_0\}$, $R_F = \emptyset$.
Induction Step: We show that, for all s and s',

$$s \in (Q_F \downarrow B) \wedge (s, s') \in (R_{IO} \downarrow B - I) \Rightarrow (s, s') \in (R_F \downarrow B - I).$$

Let $s \in Q_F \downarrow B$ and $(s, s') \in R_{IO} \downarrow B - I$. Then there is a $q \in Q_F$ such that $q \downarrow B = s$. Since $R_F \subseteq R_{IO}$, also $q \in Q_{IO}$. Since (N, q_0) is IO-deterministic, there is a q' such that $\{q'\} = out(R_a, q)$, where a is the input vector of q.

Suppose s is stable. Then all of the R_a-successors of q are in Q_{IO} because of Rule 2', and in R_F because of Rule 1. It follows from Property 4 that no changes occur in any $(R_a \downarrow B)$-sequence from q to q'. Consequently, $q' \downarrow B = s$. Since s is stable and B is simple deterministic, (s, s') must be an input transition in which only one input variable changes. Since q' is stable, we can apply Rule 2 and add $(q', q'^{\{i\}})$ to R_F, where i is the index of the input that changes, i.e., $q'^{\{i\}} \downarrow B = s'$. Consequently, $(s, s') \in R_F \downarrow B - I$.

If s is unstable then it follows by Property 4 that, in any $(R_a \downarrow B)$-sequence from q to q', there exist two consecutive states whose restrictions yield the output transition (s, s'). Since s is unstable and (N, q_0) is IO-consistent, q must also be unstable. By Rule 1, all the R_a-successors of q are in Q_F. From this it follows that $(s, s') \in R_F \downarrow B - I$. This completes the proof of the second claim. □

We say that an input-output behavior B has a *delay-insensitive* IO-realization (N, q_0) iff *all* wire delays are included as gate components of N and (N, q_0) IO-realizes B. Similarly we define a delay-insensitive F-realization of B by network (N, q_0). For example, in the network N_4 of Figure 10 all wire delays are included as gate components. Consequently, $(N_4, 0000)$ is a delay-insensitive F-realization of B_3. For a recent survey on other formalizations of delay-insensitive circuits, we refer the reader to [2].

6 A Behavior not Realizable in the Input-Output Mode

We will show in this section that the behavior B_3 of Figure 9 does not have a delay-insensitive realization operated in the input-output mode, although it has a delay-insensitive realization operated in the fundamental mode as we have seen in Example 9. We show that, if a delay-insensitive IO-realization (N, q_0) of B_3 existed, then we could construct a network N' that would have contradictory properties when operated in fundamental mode. The proof uses a rather deep result concerning the equivalence of GMW analysis and ternary simulation. Ternary simulation uses the values 0, 1, and X and a partial order on these values such that X is greater than either one of the binary values. The concept of least upper bound is defined with respect to this partial order. Ternary simulation consists of two parts, called Algorithms A and B, whose results can be characterized as follows [3](also [1] for Algorithm A):

Theorem 6 *Let N be any network in which all wire delays are taken into account explicitly. Then the result of Algorithm A of the ternary simulation of a transition of N is equal to the least upper bound of the set of states that are reachable from the initial state in the GMW analysis of N. Furthermore, the result of Algorithm B of the ternary simulation is equal to the least upper bound of the set of states that appear in the outcome of the GMW analysis.* □

For this section we assume that N has one input x that is restricted to u and one gate variable z that is restricted to v. The remaining input variables of N are assumed to be constant, and will therefore be ignored. The remaining gate variables of N are represented by vector y, and we assume that the state of N can be represented by the vector xyz.

Lemma 7 *The behavior $B_3 = \langle 1, 1, S, T, 00\rangle$, where S and T are given by*

$$B_3 : 00 \xrightarrow{u} 10 \xrightarrow{v} 11 \xrightarrow{u} 01,$$

does not have a delay-insensitive IO-realization.

Proof: If gate network (N, q_0) is a delay-insensitive IO-realization of behavior B_3, then it must have the following properties:

P_1 There exists an initial stable state $q_0 = 0b0$ for some vector b.

P_2 The state $1b0$ is unstable (since (N, q_0) is IO-consistent and 10 is unstable in B_3).

P_3 In every R_1-sequence starting with $1b0$ and ending with a state in $out(R_1, 1b0)$, z changes exactly once (Property 4).

P_4 Let $1c1$ be any state that can be reached by an R_1-sequence from $1b0$. If the input x is changed to 0 again, where some of the variables in y may also change at the same time, some state $0d1$ is reached. If the GMW analysis is continued from state $0d1$, the gate variable z should not change. Note that state $1c1$ does not have to be a stable network state.

Consider the network N' derived from N as shown in Figure 11. Notice that a delay element is introduced for every connection wire. Since network N also contains a delay element for each connection wire, Theorem 6 is applicable to network N'.

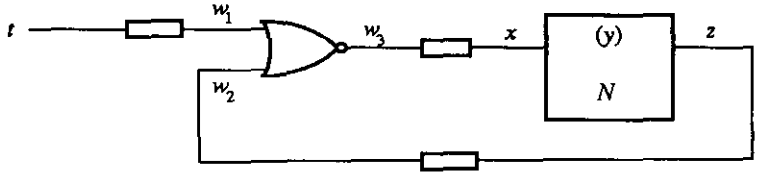

Figure 11: Network N'.

One verifies that the state $twxyz = 11000b0$ is stable. We will operate N' in fundamental mode, causing a transition by changing t from 1 to 0 and then letting the network stabilize. We have the following R_0-sequence for N':

$$01000b0 \xrightarrow{w_1} 00000b0 \xrightarrow{w_3} 00010b0 \xrightarrow{x} 00011b0.$$

Note that, until the last step above, N has been stable, as guaranteed by P_1. In the last step, N becomes unstable, as required by P_2. By P_3, N eventually reaches a state $1c1$, for some vector c, by a subsequent R_0-sequence for N', i.e.,

$$0001\underline{b0} \longrightarrow^* 0001\underline{c1}.$$

From P_4 it now follows that z cannot change any more, even if x becomes 0 again; this has to hold for all possible values that y may reach. Thus, the y-component of the state of N' becomes irrelevant, and we replace it by '_' from now on. We have the following extension of the R_0-sequence for N':

$$0001\underline{c1} \xrightarrow{w_2} 0011\_\underline{1} \xrightarrow{w_3} 0010\_\underline{1} \xrightarrow{x} 0010\_\underline{1}.$$

In the last state, the variables w, x, and z are stable and will not become unstable again. From the above, it follows that the outcome of the GMW analysis of N' started in state $0010\underline{0d1}$, where the input is kept at 0, always yields states of the form 00100_1, i.e.,

$$q \in out(R_0, 0100\underline{0b0}) \;\Rightarrow\; \text{the } z \text{ component of } q \text{ is } 1.$$

Consequently, even in the presence of arbitrary gate and wire delays, the final outcome of the transition yields $z = 1$.

We also observe that, in the above analysis, N' is operated in fundamental mode with respect to behavior

$$B' : \; 00 \xrightarrow{t} 10 \xrightarrow{z} 11,$$

but N is operated in input-output mode with respect to behavior B_3.

Next we show that ternary simulation of N' contradicts the conclusion reached above. The reader unfamiliar with ternary simulation is referred to [3]; here we only give the results of the simulation.

Algorithm A of the simulation produces the following initial sequence:

$$X1000\underline{b0} \longrightarrow XX000\underline{b0} \longrightarrow XX0X0\underline{b0} \longrightarrow XX0XX\underline{b0}$$

As we have seen above,

$$0100\underline{0b0}\; {R_0}^*\; 0100\underline{0b0} \text{ and } 0100\underline{0b0}\; {R_0}^*\; 0001\underline{1c1},$$

i.e., both $0100\underline{0b0}$ and $0001\underline{1c1}$ are reachable from $0100\underline{0b0}$ (in zero or more steps). Consequently, the output z can take the values 0 and 1 in the GMW analysis of the network. But then, by Theorem 6, Algorithm A of the ternary simulation must produce $z = X$. Subsequently, w_2 becomes X, and the final result of Algorithm A has the form $XXXXXeX$ for some vector e of ternary values.

Changing t from X to 0 now and applying Algorithm B to the state $0XXXXeX$, we find that the algorithm terminates in the second step with the state $00XXXeX$. Consequently, Algorithm B predicts that z has the value X. But then, by Theorem 6, there exists a state in the outcome of the GMW analysis where $z = 0$. This contradicts the GMW analysis above. Therefore, the network N with the postulated properties cannot exist, and we have proved

that behavior B_3 does not have a delay-insensitive gate realization operated in the input-output mode. □

With the above lemma it is easy to verify that the input-output behaviors of the set-reset latch, the C-ELEMENT, and the TOGGLE of Figures 1, 2, 3, and 4 do not have delay-insensitive realizations operated in input-output mode. Each of these behaviors contains behavior B_3 of Lemma 7. The latch has the following sub-behavior:

$$000 \xrightarrow{u_1} 100 \xrightarrow{v} 101 \xrightarrow{u_1} 001.$$

Thus, simply by ignoring the input u_2, we obtain the behavior B_3. Similarly, the C-ELEMENT has the sub-behavior:

$$100 \xrightarrow{u_2} 110 \xrightarrow{v} 111 \xrightarrow{u_2} 101.$$

If we ignore the input u_1, we obtain behavior B_3. Finally, the TOGGLE contains the behavior

$$000 \xrightarrow{u} 100 \xrightarrow{v_1} 110 \xrightarrow{u} 010 \xrightarrow{v_2} 011$$

and we obtain B_3 by ignoring the second output.

By means of slight modifications in the proof of Lemma 7, we can show that three other behaviors also lack delay-insensitive IO-realizations.

Lemma 8 *Any behavior $B = \langle 1, 1, S, T, s_0 \rangle$, where S and T are given by*

$$B: \quad a\,b \xrightarrow{u} \bar{a}\,b \xrightarrow{v} \bar{a}\,\bar{b} \xrightarrow{u} a\,\bar{b}$$

with $a, b \in \{0, 1\}$, does not have a delay-insensitive gate realization operated in input-output mode.

Proof: In case $ab = 10$, repeat the arguments of Lemma 7, but with network N' modified as follows. Insert an inverter in series with a delay in the wire leading to the input x of network N.

In case $ab = 01$, modify network N' by the addition of an inverter in series with a delay in the wire leaving output z of network N.

In case $ab = 11$, modify network N' by the addition of two inverters with delays as indicated in the two cases above. □

In the following section we show that a larger class of behaviors does not have delay-insensitive gate realizations operating in the input-output mode.

7 Nontrivial Sequential Behaviors

An example of a simple deterministic input-output behavior that does have a delay-insensitive IO-realization is shown in Figure 12. This behavior is realizable by an inverter with input u and output v. The behavior is rather trivial, however, since every input vector uniquely determines the state of the network.

$$01 \xrightarrow{u} 11 \xrightarrow{v} 10 \xrightarrow{u} 00 \xrightarrow{v} 01$$

Figure 12: Input-output behavior realized by an inverter.

Another rather trivial input-output behavior that has a delay-insensitive IO-realization is given in Figure 13, where either 00 or 01 is the initial state. This behavior can be realized by an output that is connected to a constant input and a 'dangling' external input u. Here there are two different stable states possible for each input value; however, these two states are not connected.

$$00 \xrightarrow{u} 10 \xrightarrow{u} 00 \qquad\qquad 01 \xrightarrow{u} 11 \xrightarrow{u} 01$$

Figure 13: Behavior realizable by a wire connected to a constant input.

In order to eliminate such trivial cases, we impose the following condition on *nontrivial* simple deterministic behaviors: if a behavior $B = \langle h, k, S, T, s_0 \rangle$ is nontrivial, then there exists at least one input vector $u = a$ for which there are at least two stable states ab and ab' where $b \neq b'$ and ab' is reachable from ab. Notice that the input-output behaviors of Figure 12 and 13 do not satisfy the above condition.

We have the following result:

Theorem 9 *No nontrivial simple deterministic behavior $B = \langle 1, k, S, T, s_0 \rangle$ with a binary input has a delay-insensitive gate realization operating in input-output mode.*

Proof: If B is nontrivial, then there exist stable states ab and ab' such that ab' is reachable from ab. Since $b \neq b'$, they must differ in at least one component. Without loss of generality, assume that they differ in their last component, i.e., that $b = cd$ and $b' = c'\overline{d}$, where $d \in \{0, 1\}$.

In case $ad = 00$, we can apply the following reasoning. Start in state $0c0$ which is stable. In order for $0c'1$ to be reachable from $0c0$, we must change u to 1 and then back to 0 some finite number of times. At some point in this sequence we must have a state $0e0$ where the output has not yet changed, but from which we can reach state $0f1$ with two input changes. Thus, we must have the sub-behavior:

$$0e0 \xrightarrow{u} 1e0 \longrightarrow 1fg \xrightarrow{u} 0fg \longrightarrow 0h1.$$

We can now consider two subcases.

Case 1: If $g = 1$ then the above sequence projects to the behavior B_3, if we ignore all but the first and the last components. By Lemma 7, B cannot be realized.

Case 2: $g = 0$. We now have the following projection:

$$00 \xrightarrow{u} 10 \xrightarrow{u} 00 \longrightarrow 01,$$

where the state 10 is stable in the behavior B, because the output is not changing by assumption. By Theorem 4.3 of [18], this behavior is not realizable by any network even if it is operating in fundamental mode. The basic result from [18] states that if an output does not change during the first input change, then it will not change during any number of subsequent input changes in any network that operates properly in fundamental mode and is free of static hazards in the outputs.

The other cases, where $ad = 10$ or $ad = 01$ or $ad = 11$ are all dealt with similarly using Lemma 8. □

8 Concluding Remarks

We have presented formal characterizations of the two modes of operation of a network of basic elements: the fundamental mode and the input-output mode. In doing so, we have used the General Multiple Winner Model for representing the behavior of a network of gates in connection with an input-output behavior that should be realized by that network.

We have restricted our formal characterizations in several ways. Firstly, we have confined ourselves to *simple deterministic* input-output behaviors. Accordingly, only one input or output may change at a time, and each output transition leads to a stable state.

Secondly, we have confined ourselves to networks of *gates*. Consequently, basic elements like the C-ELEMENT,TOGGLE, and ARBITER were not allowed. This choice was made, because we wanted to investigate the basic limitations of gate circuits, in view of the well-established use of gates as primitive elements for the design of synchronous circuits.

Thirdly, we have defined input-output behaviors using the concept of *state* and *transition*, where a state is uniquely represented by a vector of binary values. The behavior of a gate network has been defined in similar terms. We have used this framework because the definition of gates, GMW analysis, ternary simulation, and fundamental mode operation are all based on the representation of a state as a vector of binary values; we have done this also for reasons of simplicity. This approach, however, imposes some restrictions on the behaviors that can be specified. Therefore, it would be more desirable to define behaviors purely in terms of input and output *events*, for example, by sets of traces. This appears to be worthy of future consideration.

Our results are not limited to traditional gate networks, but apply to the more modern technologies as well. It has been shown in [4] that GMW analysis and ternary simulation are also applicable to a large class of MOS circuits. Our presentation above was limited to gate networks only for reasons of convenience.

For some related work, we refer the reader to a recent work concerning other limitations of delay-insensitive circuits [12]. That work uses a different basic formalism and treats FORKs, logic gates, and some other single-output components like C-ELEMENTS, as basic elements.

In closing, we may draw the following two conclusions from the present paper. First, if one wants to realize components like C-ELEMENTS, TOGGLEs, or latches by circuits using only gates, then one has to make some assumptions about the gate and wire delays. Second, it

follows that a set of components different from the set of logic gates is needed for the realization of a significant class of delay-insensitive behaviors. Such sets of primitive components for particular classes of delay-insensitive behaviors are suggested in [8, 22], for example.

Acknowledgement

The authors wish to thank Tom Verhoeff and Michael Yoeli for their careful reading of earlier drafts of this paper and for their many constructive suggestions.

References

[1] R.E. Bryant, *Towards a Proof of the Brzozowski-Yoeli Conjecture on Ternary Simulation*, Unpublished Manuscript, December 1983.

[2] J.A. Brzozowski and J.C. Ebergen, Recent Developments in the Design of Asynchronous Circuits, *Proceedings FCT '89*, Lecture Notes in Computer Science, Vol. 380, Springer-Verlag, pp. 78-94, August 1989.

[3] 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.

[4] J.A. Brzozowski and C-J. Seger, A Unified Framework for Race Analysis of Asynchronous Networks, *J. ACM*, Vol. 36, No. 1, pp. 20-45, January 1989.

[5] 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.

[6] T.A. Chu, *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*, PhD Thesis, MIT, 1987.

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

[8] J.C. Ebergen, *Translating Programs into Delay-Insensitive Circuits*, CWI Tract 56, Centre for Mathematics and Computing Science, Amsterdam, 1989.

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

[10] A.J. Martin et al., The Design of an Asynchronous Microprocessor, in *Advanced Research in VLSI, Proceedings of the Decennial Caltech Conference on VLSI*, (C.L. Seitz ed.), 1989.

[11] A.J. Martin, Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits, in: *UT Year of Programming Institute on Concurrent Programming*, (C.A.R. Hoare ed.), Addison-Wesley 1989.

[12] A.J. Martin, The Limitations to Delay-Insensitivity in Asynchronous Circuits, *Proc. 6th MIT Conference on Advanced Research in VLSI*, MIT Press, 1990.

[13] E.J. McCluskey, Fundamental Mode and Pulse Mode Sequential Circuits, *Proc. IFIP Congress 62*, North-Holland, pp. 725-730, 1963.

[14] T. Meng, R. Brodersen, and D. Messerschmitt, Automatic Synthesis of Asynchronous Circuits from High-Level Specifications, *IEEE Trans. on Computer Aided Design*, Vol. 8, No. 11, pp. 1185-1205, November 1989.

[15] C.E. Molnar, T.P. Fang and F.U. Rosenberger, Synthesis of Delay-Insensitive Modules, in *Proceedings 1985, Chapel Hill Conference on VLSI*, (H. Fuchs ed.), Computer Science Press, pp.67-86, 1985.

[16] D.E. Muller and W.S. Bartky, A Theory of Asynchronous Circuits, *Proceedings of an International Symposium on the Theory of Switching*, Vol. 29 of the *Annals of the Computation Laboratory of Harvard University*, Harvard University Press, Cambridge, Mass., pp. 204-243, 1959.

[17] M. Rem, Trace Theory and Systolic Computations, in *Proceedings PARLE, Parallel Architectures and Languages Europe*, Vol. 1, (J.W. de Bakker, A.J. Nijman and P.C. Treleaven eds.), Springer-Verlag, pp. 14-34, 1987.

[18] C-J. Seger, Models and Algorithms for Race Analysis in Asynchronous Circuits, Ph.D. Thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, May 1988.

[19] C.L. Seitz, System Timing, in *Introduction to VLSI Systems*, C. Mead and L. Conway, Addison-Wesley, pp. 218-262, 1980.

[20] I.E. Sutherland, Micropipelines, *Communications of the ACM*, **32** (6) (1989) pp. 720-738.

[21] C. van Berkel, C. Niessen, M. Rem, R. Saeijs, VLSI Programming and Silicon Compilation: a Novel Approach from Philips Research, *Proceedings of IEEE International Conference on Computer Design 1988, (ICCD '88)*, 1988.

[22] M. Yoeli, *Net Based Synthesis of Delay-Insensitive Circuits*, Technical Report No. 609, Department of Computer Science, Technion- Israel Institute of Technology, Haifa, Israel, February 1990.

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS--circuits.
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes.
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films.
85/04	T. Verhoeff H.M.L.J.Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate.
86/01	R. Koymans	Specifying message passing and real-time systems.
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specification of information systems.
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures.
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several systems.
86/05	J.L.G. Dietz K.M. van Hee	A framework for the conceptual modeling of discrete dynamic systems.
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP.
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers.
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987).
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language.
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing.
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86).
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes.
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4).

86/14	R. Koymans	Specifying passing systems requires extending temporal logic.
87/01	R. Gerth	On the existence of sound and complete axiomati zations of the monitor concept.
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases.
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems.
87/04	T.Verhoeff	Delay-insensitive codes - An overview.
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.
87/08	H.M.J.L. Schols	The maximum number of states after projection.
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
87/13	J.C.S.P. van der Woude	Playing with patterns - searching for strings.
87/14	J. Hooman	A compositional proof system for an occam-like real-time language.
87/15	C. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts.
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas.
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples.

87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces.
87/19	A.J.Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen.
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations.
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG.
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence.
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems.
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples.
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to Handle Structured Information.
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems.
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication.
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way.
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification.
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling.
88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits.
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases.
88/12	A.E. Eiben	Abstract theory of planning.
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees.

88/14	H.M.M. ten Eikelder R.H. Mak	Language theory of a lambda-calculus with recursive types.
88/15	R. Bos C. Hemerik	An introduction to the category theoretic solution of recursive domain equations.
88/16	C.Hemerik J.P.Katoen	Bottom-up tree acceptors.
88/17	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable specifications for discrete event systems.
88/18	K.M. van Hee P.M.P. Rambags	Discrete event systems: concepts and basic results.
88/19	D.K. Hammer K.M. van Hee	Fasering en documentatie in software engineering.
88/20	K.M. van Hee L. Somers M.Voorhoeve	EXSPECT, the functional part.
89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a paralell program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.

89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.
90/1	W.P.de Roever-H.Barringer C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, pp. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.