Testing Delay-Insensitive Circuits

Alain J. Martin
and
Pieter J. Hazewindus

Computer Science Department
California Institute of Technology

Caltech-CS-TR-90-17

Testing Delay-Insensitive Circuits

Alain J. Martin and Pieter J. Hazewindus
Computer Science Department
California Institute of Technology
Pasadena, California 91125

Abstract

We show that a single stuck-at fault in a non-redundant delay-insensitive circuit results in a transition either not taking place or firing prematurely, or both, during an execution of the circuit. A transition not taking place can be tested easily, as this always prevents a transition on a primary output from taking place. A premature firing can also be tested but the addition of testing points may be required to enforce the premature firing and to propagate the transition to a primary output. Hence all single stuck-at faults are testable. All test sequences can be generated from the high-level specification of the circuit. The circuits are hazard-free in normal operation and during the tests.

1 Introduction

A circuit is said to be delay-insensitive when its correct operation does not depend on propagation delays in wires and operators. Such a circuit is asynchronous since no clock can be used for the sequencing of operations. Traditional methods for testing synchronous circuits partition the circuit into pieces of combinational logic that can be tested separately [1, 3]. Such methods would be too expensive in terms of the size of the testing circuitry because of the high ratio of state-holding elements to combinational elements in delay-insensitive circuits.

A difficulty traditionally attached to testing delay-insensitive circuits is the problem of testing for hazards and critical races [3]. However, delay-insensitive circuits can be and should be designed without introducing races and hazards. Not only does such an approach eliminate the problem of races and hazards (which, incidentally, also plagues the testing of synchronous circuits), but it also imposes a "conservative" design style in which each transition on a variable has to be acknowledged by a transition on another variable. As we shall see, this requirement makes it possible to test single stuck-at faults simply.

In the following sections, we first define delay-insensitive circuits. Next, we introduce the *single stuck-at fault model.* We define when a fault is *testable*, and we derive a test for a circuit from the specification that the circuit implements. We show that with such a test, all faults on outputs of gates and most faults on inputs of gates are testable, as they cause the circuit

to halt. We demonstrate that the faults on inputs of gates that do not cause the circuit to halt are also testable, as they cause certain transitions to fire prematurely. Observing such premature firings may require the addition of *testing points.*

Since the testing method requires no delay assumption, the results should extend readily to other models of asynchronous circuits [2, 8, 9].

2 Delay-Insensitive Circuits

A delay-insensitive circuit is a collection of combinational and state-holding gates and their interconnections (wires and forks). Delays are arbitrary, with the exception of the delays associated with so-called *isochronic forks.* For an isochronic fork, the differences in propagation delays in the branches of the fork are assumed to be less than a gate delay. Because of this delay assumption, these circuits are sometimes referred to as quasi-delay-insensitive [7].

A gate is described as a pair of *production rules.* A production rule (PR) consists of a boolean expression (the guard) and an assignment of the output of the gate to either true or false. (A change of value of a variable as a result of an assignment is called a *transition.*) If the guard of a PR evaluates to true and remains true, then the PR *fires*, i.e., the corresponding output transition takes place. The assignment of true to z is written $z\uparrow$ and is called an up-transition; the assignment of false to z is $z\downarrow$ and is called a down-transition.

In the rest of the paper, we will assume that a guard is in disjunctive-normal form, that is, it is either a literal, a term, or a disjunction of terms. A literal is a variable or its negation; a term is a conjunction of literals.

For instance, the PRs for the NAND gate with inputs x and y, and output z are $x \wedge y \mapsto z\downarrow$ and $\neg x \vee \neg y \mapsto z\uparrow$. If $z\uparrow$ fires in a state where $\neg z$ holds, the firing is said to be *effective*; otherwise it is *vacuous.* In the sequel, we shall only consider effective firings of PRs.

We require that the guards of the PRs of a gate be mutually exclusive. This is known as *non-interference.* Also, if the guard of a PR evaluates to true, then the guard must remain true until the firing is completed. This is known as *stability.* Non-interference and stability of PRs guarantee that the circuit is free from races and hazards [6]. (Non-interference can be a property of either the gate or the circuit; stability is a property of the circuit.)

We call the input and output variables of the circuit *primary inputs* and *primary outputs* so as to distinguish them from inputs and outputs of gates. A circuit is an implementation of a *specification.*

Definition 2.1 *A specification is a set of partial orders of transitions on the primary inputs and the primary outputs of the circuit.*

We use a program notation called *handshaking expansion* to describe the specification of a circuit, but any other notation can be used. In our nota-

tion, the transitions on input and output variables are explicit assignment commands, but the circuit only "observes" the transitions on input variables by a wait action. A wait for a boolean expression B to become true is denoted $[B]$. The control structures used in the paper are the semicolon to sequence two actions, and the infinite repetition of an arbitrary program part S, denoted $*[S]$.

Example 2.1 *The following circuit, known as a D-element, is used in control circuitry to implement basic sequencing of communications [4]. It has a left channel, with input li and output lo, and a right channel, with input ri and output ro. Its specification is:*

$$*[[li]; lo\uparrow; [\neg li]; ro\uparrow; [ri]; ro\downarrow; [\neg ri]; lo\downarrow].$$

An internal variable, u, is needed in the implementation so that each state of the system can be uniquely identified in terms of the values of the variables. The specification extended with the internal variable gives:

$$*[[li]; u\uparrow; [u]; lo\uparrow; [\neg li]; ro\uparrow; [ri]; u\downarrow; [\neg u]; ro\downarrow; [\neg ri]; lo\downarrow].$$

(This extension of the specification is usually called a handshaking expansion of the circuit. There are several ways to insert the internal variable assignments in the specification.)

2.1 Environment

A circuit defined by its handshaking expansion, in general, does not constitute a complete, functional system: The primary inputs need to be set and reset by another component called the *environment* of the circuit. We describe the environment as another handshaking expansion whose primary inputs and primary outputs are the primary outputs and primary inputs of the circuit, respectively.

For instance, a possible environment for the D-element is the handshaking expansion:

$$*[li\uparrow; [lo]; li\downarrow; [ro]; ri\uparrow; [\neg ro]; ri\downarrow; [\neg lo]].$$

If the circuit is correct, there is at least one environment such that the circuit and the environment together constitute a deadlock-free, nonterminating (closed) concurrent system.

2.2 Acknowledgment Property and Isochronic Forks

In order to implement stability, the complete system consisting of a circuit and its environment must fulfill the following Acknowledgment Property.

Property 2.1 (Acknowledgment) *Each transition on the output of a gate causes a transition on the output of another gate. The second transition is the* acknowledgment *of the first one.*

Since the environment can observe only the primary outputs of the circuit, each transition on a primary input is followed by a transition on a primary output after a finite sequence of transitions on internal variables.

In this paper, we postulate that any execution of the circuit is either non-terminating or terminates with a transition on a primary output.

The Acknowledgment Property does not guarantee that each transition on an *input* is acknowledged. (In [7], it has been shown that requiring that all transitions—both input and output—be acknowledged restricts too drastically the class of circuits one can build.)

Consider a gate g with output z; g is directly connected to n, $n > 1$, gates $g1, g2, \ldots, gn$, by a fork with input z and n outputs $z1, z2, \ldots, zn$. Let $u1, u2, \ldots, un$ be the outputs of $g1, g2, \ldots, gn$, respectively.

The fork is defined by the two PRs:

$$\begin{aligned} z &\mapsto z1\uparrow, \ldots, zn\uparrow \\ \neg z &\mapsto z1\downarrow, \ldots, zn\downarrow . \end{aligned}$$

According to the Acknowledgment Property, a transition on z need only cause a transition on a single u, say $u1$. But by construction of the fork, a transition on z causes a transition on each of the zi variables. Hence, transitions on $z2, \ldots, zn$ are not acknowledged by a transition on the outputs of the gates $g2, \ldots, gn$, respectively. But we need to guarantee that the transitions on $z2, \ldots, zn$ are completed before the following transition on z takes place. We therefore introduce a delay assumption in the definition of certain forks called *isochronic forks.*

Definition 2.2 (Isochronic fork) *In an isochronic fork, the differences in delays between the branches of the same fork are small enough compared to the delays in other gates that when a transition on one of the branches of the fork has been acknowledged, we may assume that the transitions on all branches are completed.*

(A delay assumption like that of isochronic fork is necessary and sufficient to construct all circuits of interest [7].)

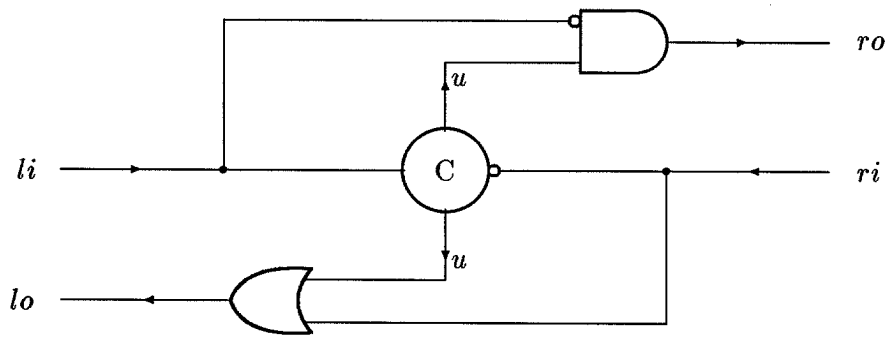

Figure 1: Circuit for the D-element

2.3 Symmetrization and Redundant Terms

Let us return to the example of the D-element. A circuit that implements the specification of the D-element has the PRs:

$$\begin{aligned} li \wedge \neg ri &\mapsto u\uparrow \\ u \vee ri &\mapsto lo\uparrow \\ \neg li \wedge u &\mapsto ro\uparrow \\ ri \wedge \neg li &\mapsto u\downarrow \\ \neg u \vee li &\mapsto ro\downarrow \\ \neg ri \wedge \neg u &\mapsto lo\downarrow \end{aligned}$$

The circuit is shown in Figure 1. For a proof that these PRs implement the specification, see [6]. Usually, for the sake of simplicity, the same variable is used in the production-rule description for the input and output of a wire and for the input and outputs of a fork, as in the above description of the D-element. This description is detailed enough to check that the Acknowledgment Property is fulfilled: We check that each assignment on the output variable of a PR is necessary for the guard of the following PR to become true.

Of particular interest are the second and fifth PRs. As they both contain a disjunction, the guard of each of them can become true as a result of either of the two terms in the disjunction becoming true. However, in the actual operation of the circuit, the guard of the second PR fires effectively only when u is true, and the guard of the fifth PR fires effectively only when u is false. The guard of the second PR could be replaced with just u and that of the fifth PR with just $\neg u$. The disjunctive form of the guards has been used in this implementation of the circuit so that the two gates can be implemented as combinational gates. This transformation is called *symmetrization.*

Symmetrization is the following transformation. Consider the two PRs

$$\begin{aligned} x \wedge y &\mapsto z\uparrow \\ \neg x &\mapsto z\downarrow \end{aligned}$$

as part of a PR set implementing a specification. If we can prove that in any state in which the second PR can fire—i.e. in any state where $\neg x \wedge z$ holds—y also holds, then we can safely replace guard $\neg x$ with $\neg x \vee \neg y$. Indeed $\neg x \vee \neg y \equiv \neg x$ in all states where the PR can fire. The term (here $\neg y$) thus added is called a *redundant term* since it never causes the PR to fire, and the transitions $y\downarrow$ are said to be *silent* since they are never acknowledged.

2.4 Production-Rule Representation with Input Variables

In order to analyze which transitions on input of gates are acknowledged, we have to refine the PR representation of the circuits. For instance, the circuit for the D-element contains three forks: the fork with input li and outputs $l1$ and $l2$, the fork with input ri and the outputs $r1$ and $r2$, and the output u of the C-element forking to the AND-gate with input $u1$ and to the or-gate with input $u2$. We extend the PR representation so as to include transitions on the branches of the forks:

$$\begin{aligned}
li &\mapsto l1\uparrow, l2\uparrow \\
l2 \wedge \neg r1 &\mapsto u\uparrow \\
u &\mapsto u1\uparrow, u2\uparrow \\
u1 \vee r2 &\mapsto lo\uparrow \\
\neg li &\mapsto l1\downarrow, l2\downarrow \\
\neg l1 \wedge u2 &\mapsto ro\uparrow \\
ri &\mapsto r1\uparrow, r2\uparrow \\
r1 \wedge \neg l2 &\mapsto u\downarrow \\
\neg u &\mapsto u1\downarrow, u2\downarrow \\
\neg u2 \vee l1 &\mapsto ro\downarrow \\
\neg ri &\mapsto r1\downarrow, r2\downarrow \\
\neg r2 \wedge \neg u1 &\mapsto lo\downarrow
\end{aligned}$$

We can now check that all transitions on inputs are acknowledged, except transitions $l1\uparrow$ and transitions $r2\uparrow$, which are never acknowledged. Hence the fork with input li and the fork with input ri have to be isochronic, and the up-transitions on $l1$ and the up-transitions on $r2$ are silent.

Observe that an isochronic fork need not have silent transitions on one of its branches. It is often the case, in particular with circuits that handle data, that for a certain value of the data, one branch of an isochronic fork has an acknowledged transition, and for another value of the data another branch of the fork has an acknowledged transition. However, the fork still requires the isochronicity assumption, to guarantee stability.

2.5 Non-redundancy

We assume that circuits are not redundant.

Definition 2.3 *Let circuit C be an implementation of specification S. A gate in C is redundant if the circuit still implements S after replacing an*

input of the gate with either **true** *or* **false**. *Circuit C is redundant if there is a redundant gate in C.*

The circuit for the D-element derived above is not redundant although it contains PRs with redundant terms. Hence, a circuit containing PRs with redundant terms need not be redundant.

3 Testing for Stuck-at Faults

3.1 The Stuck-at Fault Model

The fault model we use is the *single stuck-at fault model.* A faulty circuit is modeled as having a single static fault. A variable can be either stuck-at-0 (permanently at a low voltage) or stuck-at-1 (permanently at a high voltage). For a fork, each branch can be stuck-at-1 or stuck-at-0 independently of the other branches and the input of the fork. Hence, a fork branching to n gates ($n > 1$) contributes $2(n + 1)$ different single stuck-at faults [1, 3].

The following example shows that a fault on the input of a fork and the same fault on one of the outputs of the same fork do not lead to the same circuit.

Example 3.1 *Consider the circuit for the D-element. If the output u of the C-element is stuck-at-0, then we replace each occurrence of u by* **false** *in the PR set and reduce boolean expressions:*

$$\begin{aligned} ri &\mapsto lo\uparrow \\ \neg ri &\mapsto lo\downarrow \end{aligned}$$

In addition, ro and u are both permanently **false**.

If input u of the gate with output ro is stuck-at-0, then replace u with **false** *in the PRs for ro:*

$$\begin{aligned} li \wedge \neg ri &\mapsto u\uparrow \\ u \vee ri &\mapsto lo\uparrow \\ ri \wedge \neg li &\mapsto u\downarrow \\ \neg ri \wedge \neg u &\mapsto lo\downarrow \end{aligned}$$

Output ro is permanently **false**. *Hence, the two faults do not result in identical circuits.*

3.2 Inhibited Transitions and Premature Transitions

The effect of a stuck-at fault on the output of a gate is straightforward. Let z be the output of a gate. If z is stuck-at-1, no transition $z\downarrow$ can take place. We say that z stuck-at-1 *inhibits* all transitions $z\downarrow$, and similarly, z stuck-at-0 *inhibits* all transitions $z\uparrow$.

But the effects that a stuck-at fault on the input of a gate has on the output of the same gate are more complex. Such a fault may inhibit transitions on the output, and/or cause transitions on the output to become enabled in an illegal state. In the latter case, we say that the transitions are *prematurely enabled.* If a transition indeed fires in an illegal state, we say that it *fires prematurely.*

Theorem 3.1 *Let x be an input variable of a gate, g. Then, at least one of the following two cases holds (B is an arbitrary boolean expression).*

1) $x \wedge B$ is a non-redundant term in the guard of a PR of g. In this case, x stuck-at-1 causes some transitions on the output of g to be prematurely enabled, and x stuck-at-0 causes some transitions on the output of g to be inhibited.

2) $\neg x \wedge B$ is a non-redundant term in the guard of a PR of g. In this case, x stuck-at-0 causes some transitions on the output of g to be prematurely enabled, and x stuck-at-1 causes some transitions on the output of g to be inhibited.

Example 3.2 *Consider the C-element defined by the PRs:*

$$\begin{aligned} x \wedge y &\mapsto z\uparrow \\ \neg x \wedge \neg y &\mapsto z\downarrow \end{aligned}$$

x stuck-at-0 causes all $z\uparrow$ transitions to be inhibited, and all $z\downarrow$ transitions to be prematurely enabled. x stuck-at-1 causes all $z\downarrow$ transitions to be inhibited, and all $z\uparrow$ transitions to be prematurely enabled.

Consider the AND gate defined by the PRs:

$$\begin{aligned} x \wedge y &\mapsto z\uparrow \\ \neg x \vee \neg y &\mapsto z\downarrow \end{aligned}$$

x stuck-at-0 causes all $z\uparrow$ transitions to be inhibited, and some of the $z\downarrow$ transitions to be prematurely enabled if the term $\neg x$ is not redundant.

x stuck-at-1 causes all $z\uparrow$ transitions to be prematurely enabled, and some $z\downarrow$ transitions to be inhibited if the term $\neg x$ is not redundant.

Observe that, if the term $\neg x$ is redundant, then x stuck-at-1 will not cause any transition to be inhibited. Hence the circuit may not halt as the result of such a fault.

Proof: Consider a gate g with input x and output z. The most general form for the PRs of g is:

$$\begin{aligned} (x \wedge B0) \vee (\neg x \wedge B1) \vee B2 &\mapsto z\uparrow \\ (x \wedge C0) \vee (\neg x \wedge C1) \vee C2 &\mapsto z\downarrow \end{aligned}$$

where the B and C expressions do not contain x.

Some of the terms may be redundant, if they have been added for symmetrization. But, since the circuit is not redundant, at least one term containing x is not redundant, for instance $x \wedge B0$, and $x \wedge B0 \not\equiv B0$ in the states where $\neg z$ holds. Hence, there is at least one state, say S, in the computation where $\neg z$ is true, and $x \wedge B0$ is true.

We first show that $x \wedge B0$ is the only true term in S. The terms of the other PR are not true because of the non-interference requirement; $\neg x \wedge B1$ is obviously not true since x is true; and $B2$ is not true either: $B2$ also holding would violate either the non-redundancy of the term $x \wedge B0$ or the stability of the PR.

Because of stability, the PR fires in S in the correct circuit. Hence, if x is stuck-at-0, the guard reduces to false, and the firing of PR cannot take place, i.e., those transitions $z\uparrow$ that fire in S in the correct circuit are inhibited in the faulty circuit.

If x is stuck-at-1, $x \wedge B0$ reduces to $B0$. Then, necessarily, the number of states in which the PR can fire has been increased. Otherwise, we would have $B0 \equiv x \wedge B0$ in all states where $\neg z$ holds, which we have excluded. Hence, there is a state in which the transition of the PR is enabled prematurely. □

Hence, any fault in a non-redundant circuit results in an incorrect circuit, since a transition is either inhibited or prematurely enabled. The testing problem is to find an environment in which the faulty circuit behaves differently from the correct one, i.e., causes a sequence of transitions on the primary inputs and outputs that violates the specification.

3.3 Tests and Testability

Definition 3.1 *A test of a circuit C is a finite execution of an environment of C.*

In order to detect a stuck-at fault on each variable of C, the execution of C with the test must cause an up- and a down-transition on each variable in C.

Definition 3.2 *Let circuit C implement specification S, and let C' be identical to C, except that C' has a single stuck-at fault. This fault is* detected *by a test T if the execution of C' together with T causes a sequence of primary output transitions that is not consistent with S, i.e., either a primary output transition fires prematurely or it is inhibited.*

When a transition on a primary output never takes place where such a transition is expected in the specification, the circuit, or a part of it, has halted indefinitely.

For example, a test for the D-element is $li\uparrow; [lo]; li\downarrow; [ro]; ri\uparrow; [\neg ro]; ri\downarrow; [\neg lo]$. Repeating this sequence any number of times is also a test. Observe that the test is indeed a prefix of the environment for the D-element described earlier.

In the D-element, if a transition $ro\uparrow$ takes place after input transition $li\uparrow$, then a fault has been detected. If, after transition $li\uparrow$, we can decide that transition $lo\uparrow$ will not occur, then we have also detected a fault.

Definition 3.3 *Consider a circuit that has one stuck-at fault. This fault is* testable *if a test can be constructed such that any execution of the circuit with the test detects the fault.*

Since propagation delays are, in theory, finite but unbounded, a premature transition may take an arbitrary time to fire, and it cannot be determined in any finite amount of time that a circuit has halted.

In practice, however, there is an upper bound on propagation delays for any chip technology. We can therefore safely assume that a transition will not occur if it has not occurred within a certain period of time. Similarly, we can assume that if an output change will take place eventually, it will take place within a certain period of time.

3.4 Non-interference and Stability in the Presence of Faults

A correct circuit has non-interfering and stable PRs. In the presence of a stuck-at fault, however, this may no longer be the case. For interfering PRs there is a simple transformation of the PRs that results in non-interference. Let z be a variable in a circuit, with PRs:

$$\begin{cases} B_0 & \mapsto & z\uparrow \\ B_1 & \mapsto & z\downarrow, \end{cases}$$

where B_0 and B_1 are arbitrary boolean expressions. If $\neg B_0 \vee \neg B_1$ is a tautology, then these two PRs are non-interfering, even in the presence of a stuck-at fault. This holds for any combinational gate, the C-element, and most generalizations of the C-element.

If an input of a flip-flop has a stuck-at fault, however, the resulting PRs may be interfering. Consider the flip-flop with PRs:

$$\begin{cases} x & \mapsto & z\uparrow \\ y & \mapsto & z\downarrow. \end{cases}$$

If input x is stuck-at-1, the first rule becomes $\mathbf{true} \mapsto z\uparrow$. If, during a test, y holds, then both $z\uparrow$ and $z\downarrow$ can fire simultaneously.

If $\neg B_0 \vee \neg B_1$ is not a tautology, we redesign the circuit by strengthening the PRs, so that the resulting production rules are non-interfering, even in the presence of a fault. For the flip-flop above we can change either the first PR to $x \wedge \neg y \mapsto z\uparrow$, or the second one to $y \wedge \neg x \mapsto z\downarrow$, or both.

A fault that causes a PR to fire prematurely may violate the stability of a PR, possibly causing some hazards. It is therefore necessary that the first premature firing caused by a fault be detected. In the presence of a fault that

causes both a premature firing and an inhibited firing, one should not rely on the detection of the inhibited firing if the premature firing happens first, even though, as we shall next see, detecting an inhibited firing is simpler.

We now turn to the problem of detecting both types of malfunctioning—inhibited firing and premature firing. We first show how an inhibited firing leads to the circuit halting during any test. We assume that no premature firing takes place during the test.

4 Faults Leading to a Circuit Halting

In this section we show that a circuit halts as the result of a single stuck-at fault causing an inhibited transition. Hence, any such fault is testable.

We assume that a non-redundant circuit contains an up-transition and a down-transition on each variable of the circuit. One transition can be part of the reset procedure. We first prove the following lemma.

Lemma 4.1 *Each transition on an internal variable is followed by a finite sequence of transitions ending with a transition on a primary output.*

Proof: Let $z1$ either be the output of a gate of the circuit, or be a primary input, i.e., the output of a gate of the environment. According to the Acknowledgment Property, each transition on $z1$ is acknowledged by a transition on the output $z2$ of another gate, say g; $z1$ is connected by a wire or a fork to an input of g.

A transition on $z2$ is also acknowledged by a transition on the output of another gate. Hence, since the closed circuit is finite, each transition on the output of a gate is included in a cycle of transitions on gate outputs, each acknowledging the previous one in the cycle.

But the circuit is non-redundant and an execution of the circuit is either a non-terminating computation consisting of alternations of transitions on primary inputs and primary outputs or terminates with a primary output transition. Hence, any cycle of transitions contains at least one transition on a primary output of the circuit. □

Theorem 4.2 *If a non-redundant circuit contains inhibited transitions on the output of a gate or on a primary input, then some transition on a primary output which takes place in the execution of the correct circuit does not take place in the execution of the faulty circuit with* any *test that exercises the inhibited transitions.*

Proof: Let s be the variable, either gate output or primary input, and let ts be the transition on s inhibited by the fault.

If s is also a primary output, then a transition on a primary output does not take place, namely, ts. If s is not a primary output, then, by the lemma above, there is a finite chain of transitions following ts and ending with a transition on a primary output.

There is a gate output z such that ts is acknowledged with a transition tz on z, and tz follows ts in the chain. Since tz acknowledges ts, and since the circuit is not redundant, if ts does not occur during the test, tz does not occur. The situation is equivalent to having a stuck-at fault on gate output z. By induction on the length of the chain, all transitions in the chain are inhibited, in particular the transition on the primary output. □

5 Testing Premature Firings

Two conditions must hold to make a fault leading to the premature firing of a PR testable. First, the circuit has to be held in a state where the PR will fire prematurely. Second, this firing has to cause a sequence of transitions that will lead to a transition on a primary output, so that the fault can be detected.

The only way to hold the circuit in a state where the premature firing will eventually take place is by postponing a transition on a primary input, since primary input transitions are the only transitions that the test can control.

Hence the set of states in which the transition can misfire has to contain a transition on a primary input. Otherwise, one of the variables that change in one of these states has to be made into a primary input.

Example 5.1 *A test for the D-element is*

$$li\uparrow;[lo];li\downarrow;[ro];ri\uparrow;[\neg ro];ri\downarrow;[\neg lo].$$

We have seen that, with a fault ri stuck-at-0, PR $\neg ri \wedge \neg u \mapsto lo\downarrow$ *will misfire if there is a state in the handshaking expansion where* $ri \wedge \neg u \wedge lo$ *holds. Such a state is the state in the handshaking expansion where the circuit waits for* $\neg ri$ *to hold. As lo is a primary output, this fault is testable if the test withholds transition* $ri\downarrow$.

Similarly, fault li stuck-at-0 can cause PR $\neg li \wedge u \mapsto ro\uparrow$ *to misfire if there is a state where* $li \wedge u \wedge \neg ro$ *holds, which is the state when the circuit waits for* $\neg li$ *to hold. As ro is a primary output, this fault is testable if the test withholds transition* $li\downarrow$.

6 Testing with Control Points

For some circuits, it is not possible to withhold a transition so that a PR will fire prematurely. Consider the case of two D-elements, where the R channel of the first element is connected to the L channel of the second (see figure 2). The handshaking expansion, including transitions of internal variables, is:

$$*[\ [l1i];u1\uparrow;[u1];l1o\uparrow;[\neg l1i];r1o\uparrow;[l2i];u2\uparrow;[u2];l2o\uparrow;[r1i];u1\downarrow;[\neg u1];\\ r1o\downarrow;[\neg l2i];r2o\uparrow;[r2i];u2\downarrow;[\neg u2];r2o\downarrow;[\neg r2i];l2o\downarrow;[\neg r1i];l1o\downarrow],$$

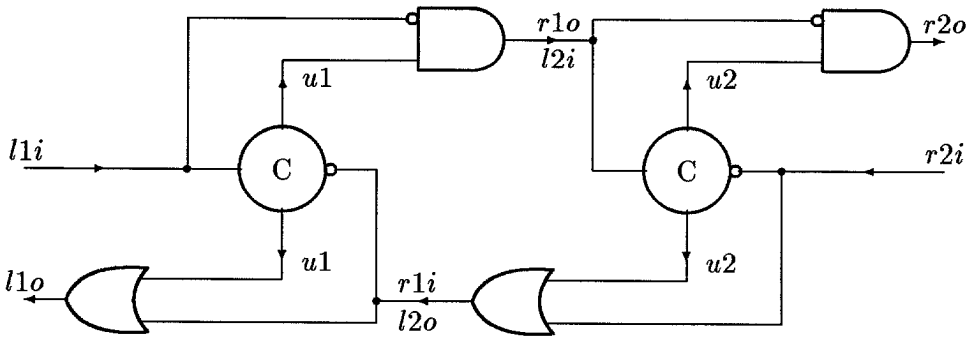

Figure 2: Circuit for two connected D-elements

The primary inputs of this circuit are $l1i$ and $r2i$, the primary outputs are $l1o$ and $r2o$. With this handshaking expansion, we construct test

$$l1i\uparrow; [l1o]; l1i\downarrow; [r2o]; r2i\uparrow; [\neg r2o]; r2i\downarrow; [\neg l1o].$$

As seen previously, fault $l2i$ stuck-at-0 can cause $r2o\uparrow$ to fire prematurely in the states going from transition $u2\uparrow$ until transition $r1o\downarrow$ in the handshaking expansion. But this part of the handshaking expansion contains no primary input change; therefore the environment cannot keep the circuit in this state for an indefinite time so as to guarantee misfiring of the PR. The fault can be made testable if $r1i$ is transformed into a primary input.

A more realistic example can be found in the FETCH process of the Caltech Asynchronous Microprocessor [5]. Part of the handshaking expansion is:

$$\ldots; [ai \wedge \neg ci]; co\uparrow; u\downarrow; [\neg u \wedge ci]; bo\uparrow; co\downarrow; ao\downarrow; [bi]; bo\downarrow; \ldots,$$

where u is a state variable that forks to several gates, and ao and ai are channel variables, as are bo and bi, and co and ci. For variable bo, the PRs are:

$$\begin{cases} \neg u \wedge ci \wedge co & \mapsto & bo\uparrow \\ \neg ao \wedge bi & \mapsto & bo\downarrow . \end{cases}$$

Consider input u stuck-at-0 for this gate. The fault is testable if the environment can hold the circuit in a state where $u \wedge ci \wedge co \wedge \neg bo$ holds, for an arbitrary amount of time, to guarantee that $bo\uparrow$ misfires. The only two transitions of co are shown above. Since a transition $u\downarrow$ directly follows $co\uparrow$, the environment cannot hold the circuit in a state where $bo\uparrow$ can misfire for an arbitrary length of time. Therefore the fault is untestable.

To make fault u stuck-at-0 testable, we replace state variable u with a channel U. Each $u\uparrow$ transition is replaced by $[ui]; uo\uparrow$, each $u\downarrow$ transition is replaced by $[\neg ui]; uo\downarrow$, and each occurrence of u in the guard of PRs is replaced with ui. Now the environment can hold the circuit in a state where $bo\uparrow$ misfires, making the fault testable. Channel U is used only during the testing of the circuit. During normal operation ui is connected directly to uo.

7 Concluding Remarks

The concluding remarks of this preliminary investigation of delay-insensitive circuit testing can be put in the "bad-news/good-news" form. The bad news is that, contrary to common belief, a single stuck-at fault does not always lead to the circuit halting, but may also lead to some transitions firing prematurely. The good news is that, also contrary to common belief, any single stuck-at fault can be tested. Testing premature firings may require adding control points to the original circuit. We are redesigning the control part of the Caltech Asynchronous Microprocessor so as to make the circuit fully testable. A first redesign indicates that the number of extra testing points is very small. Three of the five processes that constitute the control of the microprocessor each require one testing point. An extra testing point is required when the processes are connected together. We have not discussed how the test sequences should be generated. They all can be generated from the high-level specification of the circuit, but that may be too expensive—in terms of the size of the vectors—for certain circuits, e.g., datapaths. We have also ignored some complications created by the reset procedures.

Acknowledgments

We wish to thank Steve Burns for many valuable contributions, and Drazen Borkovic, Marcel van der Goot, Tony Lee, and Jose Tierno for their comments on earlier versions of the manuscript. The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202; and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

References

[1] M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press (1990)

[2] T.A. Chu, *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*, Ph.D. Thesis, MIT (1987)

[3] H. Fujiwara, *Logic Testing and Design for Testability*, MIT Press (1985)

[4] A.J. Martin, "Self-timed FIFO: An Exercise in Compiling Programs into VLSI Circuits," *Conference Proceedings, From HDL Descriptions to Guaranteed Correct Circuit Designs*, ed. D. Borrione (1986)

[5] A.J. Martin et al., "The Design of an Asynchronous Microprocessor," *Proceedings, Decennial Caltech Conference on VLSI*, ed. C.L. Seitz, MIT Press, pp. 351–371 (1989)

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

[7] A.J. Martin, "The Limitations to Delay-insensitivity in Asynchronous Circuits," *Proceedings, Sixth MIT Conference on Advanced Research in VLSI*, MIT Press (1990)

[8] T.H.-Y. Meng, R.W. Brodersen, and D.G. Messerschmitt, "Automatic Synthesis of Asynchronous Circuits from High-Level Specifications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 8 (11), pp. 1185–1205 (1989)

[9] J. Staunstrup and M.R. Greenstreet, "Synchronized Transitions," in *Formal Methods for VLSI Design*, ed. J. Staunstrup, Elsevier Science Publishers B.V. (1990)

[10] M.J.Y. Williams and J.B. Angell, "Enhancing Testability of Large Scale Integrated Circuits Via Test Points and Additional Logic," *IEEE Transactions on Computers*, vol. C-22, pp. 46–60 (January 1973)