Copyright © 1989, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

EXTENDED STUCK-FAULT TESTABILITY FOR COMBINATIONAL NETWORKS

by

Patrick C. McGeer, Robert K. Brayton, Richard L. Rudell,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M89/87

21 July 1989

COVER PAGE

EXTENDED STUCK-FAULT TESTABILITY FOR COMBINATIONAL NETWORKS

by

Patrick C. McGeer, Robert K. Brayton, Richard L. Rudell,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M89/87

21 July 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

EXTENDED STUCK-FAULT TESTABILITY FOR COMBINATIONAL NETWORKS

by

Patrick C. McGeer, Robert K. Brayton, Richard L. Rudell,
and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M89/87

21 July 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Extended Stuck-Fault Testability for Combinational Networks

Patrick C. McGeer, Robert K. Brayton, Richard L. Rudell*
Alberto L. Sangiovanni-Vincentelli

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley†

Abstract

We argue that a synchronous logic circuit computing a vector boolean function $\mathbf{f}(\mathbf{x})$ does so in some time T, and hence the relevant value of the circuit is not its static value (the value at $t = \infty$) but rather its value at $t = T$. We demonstrate a circuit containing an untestable stuck-at-0 fault with the property that, when the fault is set, the delay of the circuit increases dramatically, and, hence, the value of the output of the circuit at $t = T$ is incorrect. We show that this fault is not a *delay fault* in the sense of [5,13,6,12], but is rather a classic stuck fault in the sense of testing theory. This fault is therefore *not* redundant by any reasonable definition, even though it is redundant by the conventional definition. We introduce a new concept of redundancy, called *τ-redundancy*. We propose a method of generating tests for τ-irredundant stuck faults, and discuss several methods of observing these faults.

1 False Paths in Combinational Logic Circuits

One issue which arises in the design of integrated circuits is that of ensuring that a design meets a set of timing constraints. Circuit simulators such as SPICE are occasionally used to make this determination; however, circuit simulation is typically too slow to be used for an entire circuit. Hence, a

*Synopsys Incorporated, Mountain View, CA
†This research supported by the Semiconductor Research Corporation under Contract 87-DC-008

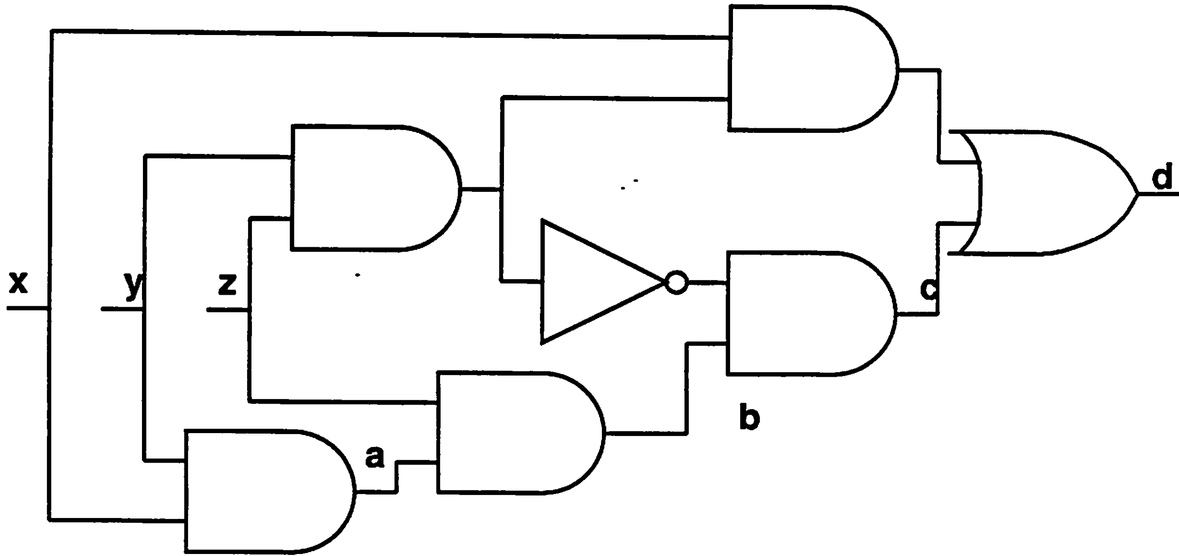


Figure 1: A False Path

common approach is the use of programs called *timing verifiers* such as [17] or [10], which are used either alone to estimate the critical delay, or to identify the critical path for later analysis by a circuit simulator. Timing verifiers are typically quite fast; indeed, for fully-restoring combinational logic the problem is simply that of finding the longest path through a directed acyclic graph, which is well known to be $O(|V|+|E|)$. However, these programs will always identify the longest path as the critical path of the circuit. This path, however, is not the path of real interest: the path of interest is the longest path down which a signal can propagate. Paths down which no signal can propagate are called *false paths*, and the problem of identifying them, and so finding the longest true path through the circuit, is known as the *false path problem.*

Consider, for example, the circuit in figure 1. For x to propagate to a, we must have $y = 1$. For a to propagate to b, we must have $z = 1$. But for b to propagate to c, we must have $y = z = 0$. Hence the path $\{x, a, b, c, d\}$ is *false.*

The false path problem has been known for some time. The earliest complete discussion in the literature appears to be due to Hrapcenko [9] [1].

[1] Hrapcenko's manuscript was kindly brought to the attention of the authors by Prof.

Hrapcenko demonstrated that, for every integer n, there exists a function for which the actual delay of the minimal network is $n + 8$ but for which the longest path is $2n + 8$. Hrapcenko further observed that false paths arise naturally in the design of carry-acceleration adders, and suggested that the longest path through a carry-acceleration adder will be on the order of $2n$ nodes, while the delay will grow approximately as n. The problem of detecting false paths for the purpose of timing verification has been extensively studied over the last few years[4,3,15,14]. A detailed treatment of the phenomenon and rigorous results can be found in [15], wherein a tight, correct, and robust algorithm to report the longest true path in a network. Hence the delay of a network may be generally assumed to be set to that of the longest sensitizable path.

2 Carry-bypass Adders and τ-Irredundant Faults

One can picture an arbitrary piece of synchronous circuitry as a set of banks of combinational logic separated by clocked registers, as in figure 2. In such a circuit, the clocks may be taken to be set to the length of the longest true path in the preceding bank of circuitry; the *effective value* of the vector boolean function $\mathbf{C_1}$ is therefore the values present at the register inputs when the clock φ_1 falls at $t = \tau_1$. From the example of figure 1, it is easy to see that a stuck fault can make a false path true.

Consider, for example, the circuit pictured in figure 3. If the marked fault is stuck-at-1, then the false path x-a-b-c-d is in fact true. Now, let x arrive at $t = 1$ while y and z arrive at $t = 0$. If x is toggled from 1 to 0 at $t = 1$, the output of the circuit in the presence of the fault does not toggle to 0 until $t = 5$, while a standard false path analysis indicates that the true critical path of the circuit is of length 4. If the clock is set to fall at $t = 4$, the value of the circuit will be incorrect at that time, even though $\bar{x}yz$ is not a test vector for the marked stuck fault (this fault is conventionally untestable: the lead is unobservable).

While the circuit of figure 3 is obviously conjured, the problem of these untestable faults occurs in real circuitry. In fact, these faults occur in circuits *which typically form the critical path of a microprocessor.* The reason for this is simply that adding circuitry to make critical paths false is a designers' favourite trick for speeding up a circuit.

N. Pippenger of UBC

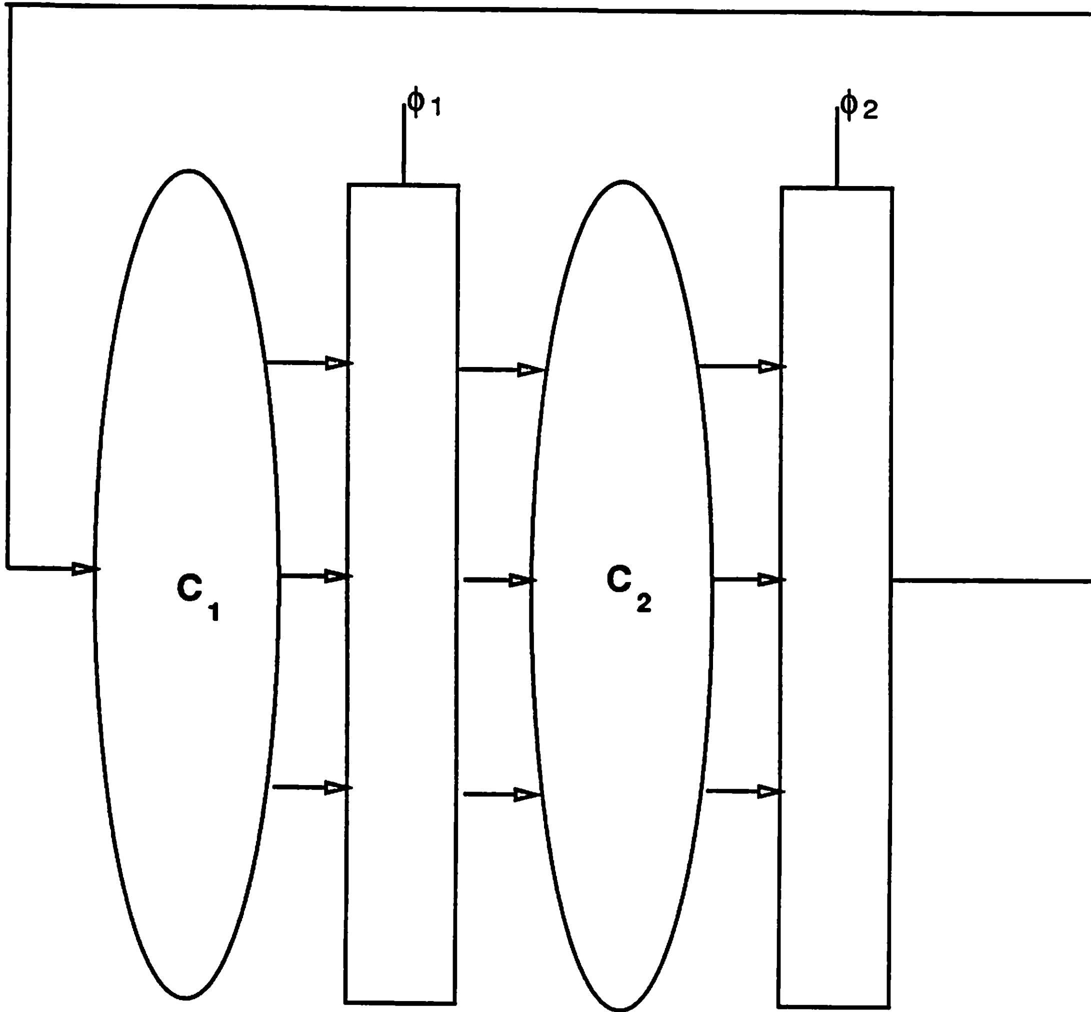


Figure 2: Generic Synchronous Circuitry

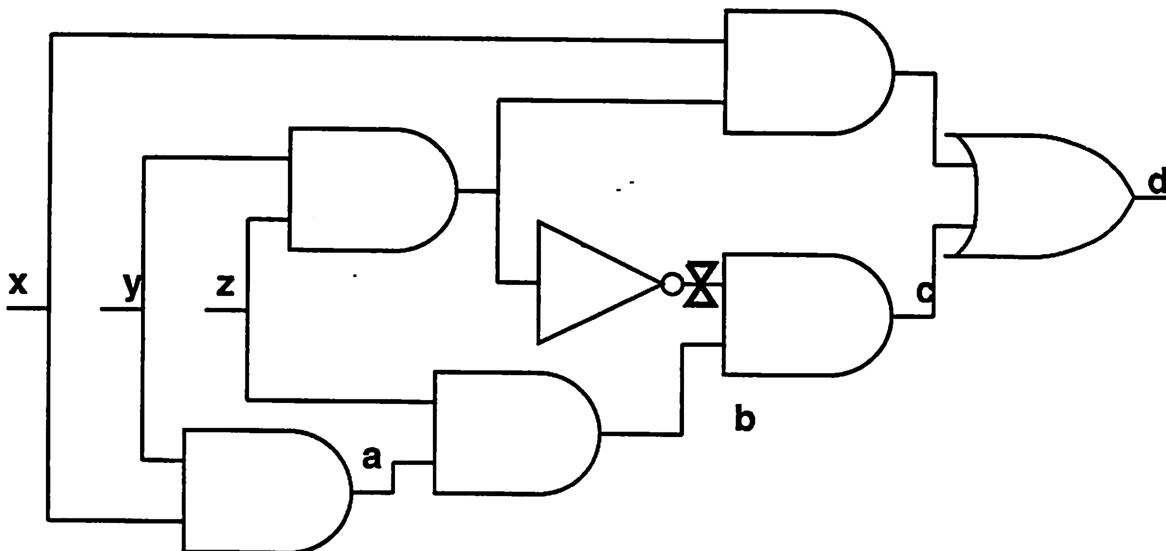


Figure 3: Stuck Fault Making Long Path True

Consider, for example, a carry-bypass adder, of which one block is depicted in figure 4. The carry-bypass adder is a conventional ripple-carry adder, with an extra AND gate and MUX added to each block, as depicted in figure 4. The circuitry derives from the simple observation that if all the propagate bits through a block are high, then the carry out of the block C_{out} is equal to the carry in to the block, C_{in}. Hence the multiplexer simply selects $C_{out} = C_{in}$ when all the propagate bits are high, and the carry chain bypasses the entire block.

The heuristic appears simple, but it has the effect of reducing the length of the true critical path by a factor of about two, depending upon the number of bits in the adder and the size of each block. The true critical path and longest path in a 16-bit carry-bypass adder with block size 4 are shown in figure 5. This diagram is adapted from one appearing in [3].

Now, consider again the picture of the carry-bypass block, shown in figure 4. Note that the observability condition for the output of the AND gate at C_{out} is $C_{b4} \oplus C_{in}$. The controllability condition to test the condition that the output of the AND gate is stuck-at-0 is $P_0 P_1 P_2 P_3$. But in that case, we must have that $C_{b4} = C_{in}$, since all the propagate lines are driven high. Hence the output of the AND gate is untestable for stuck-at-0 in the

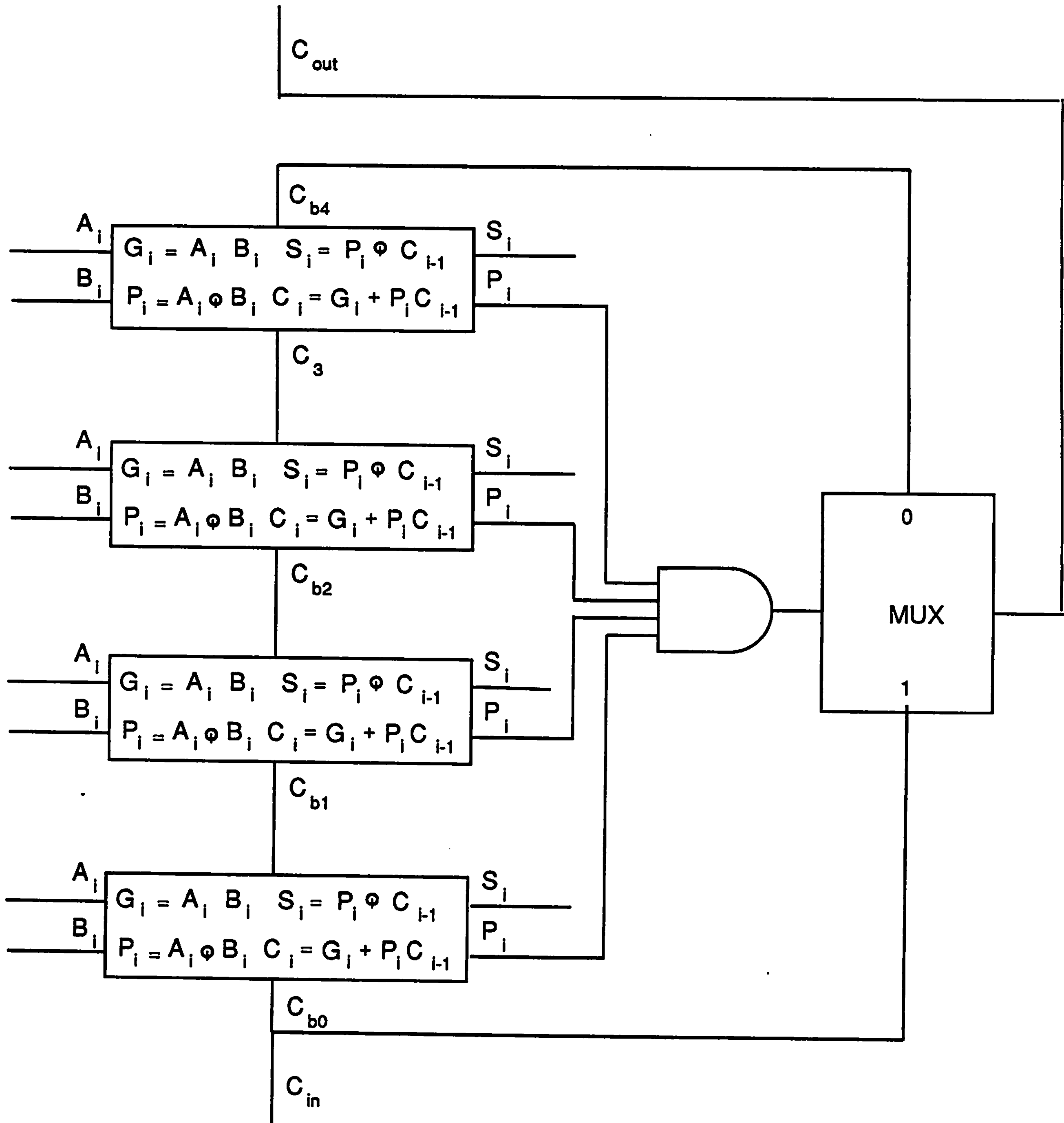
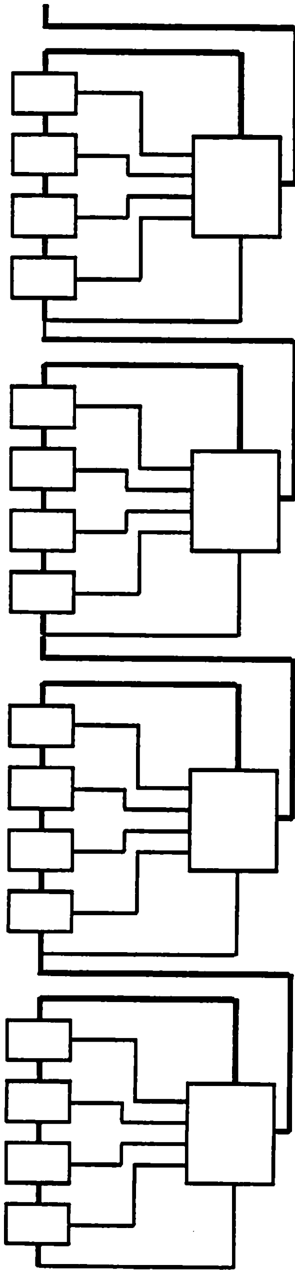
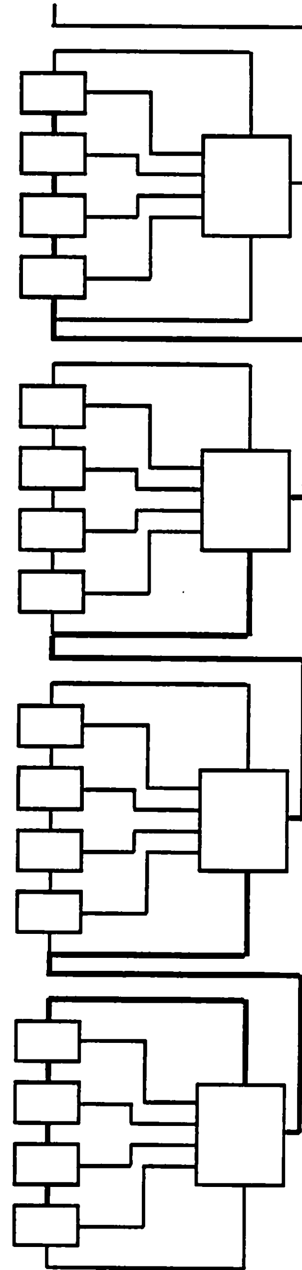


Figure 4: Four Bit Block of a Carry-Bypass Adder



Longest Path Through Adder



Longest True Path Through Adder

Figure 5: Critical Paths in a Carry-Bypass Adder

conventional sense.

Nevertheless, consider the case where $a_i \neq b_i$ for every i, i.e., the output of each AND gate is controlled high. In this case, a stuck-at-0 fault on the output of one of the AND gates will increase the length of the critical path by approximately 4. If the clock is set to close at the length of the critical path as reported by a true critical path algorithm such as [14,15], as is likely in practice, then the output of the circuit will be in error when the register is closed. In other words, the "untestable" fault caused the circuit to behave incorrectly.

Note that this fault is *not* a delay fault in the sense of [5,13,6,12]. That theory was concerned with determining the behaviour of a circuit where each gate was performing correctly, but where some gates switched slower than nominally. This fault is a classic stuck fault that demonstrates its effects in the time domain, and must be attacked by a combination of classic stuck-fault testing techniques and the new algorithms devised recently for timing verification.

This example has far-reaching consequences for the fields of logic synthesis and test generation. It has long been conjectured in the logic synthesis community [2] that untestable faults could easily be removed from a circuit, once found. This is a central focus of recent improvements to the BOLD synthesis system [?]. Further, it has recently been stated that [?] 100% stuck-fault and multifault testability is a desirable goal of logic synthesis. This example indicates that such a goal may well be undesirable, unless it can be shown that each such circuit can be transformed into a 100% testable circuit with no impact on area or delay. We conjecture that this will never be shown; we believe that it is not true, though this is very much an open question.

The test generation community must also consider such faults. The stuck fault raised here can lead to catastrophic failure of the circuit, and hence it is of high priority to devise means of testing and observing this fault. We shall see below that observing this fault is problematic in general circuits.

The next three sections are taken virtually verbatim from [15]. Readers familiar with the contents of that paper may wish to skip these sections, and proceed directly to section 6.

3 Basic Definitions, Notation, and Dynamic Sensitization

In this paper, we will be computing the conditions under which paths may transmit events as the satisfying sets of logic functions. Some elementary definitions from the theory of logic functions follow.

Definition 3.1 *A* **cube** *is a product of literals; eg,* xyz, $\overline{x}y$

Definition 3.2 *The* **cofactor** *of a function* f *wrt a literal* x ($\overline{x}$), *written* f_x ($f_{\overline{x}}$), *is the function obtained by evaluating* f *at* $x = 1$ ($x = 0$). *Since it is clear that for any literals* x, y, $(f_x)_y = (f_y)_x$, *we write this as* f_{xy} *and so define the cofactor of* f *over any cube* $c = x_1 ... x_n$ *in the obvious way:* $f_c = f_{x_1 ... x_n}$.

This definition yields a classic theorem, due to Shannon:

Theorem 3.1 (Shannon Expansion) *Given any function* f, *any variable* x, *we have:*

$$f = x f_x + \overline{x} f_{\overline{x}}$$

For purposes of clarity, we outline a very simple timing model here. The results of this paper, however, do not depend on the precise characteristics of this model; we can show that they hold for slope delay models, models with separate rise and fall delays, and different delays on each pin.

Definition 3.3 *A* **path** *through a combinational circuit is a sequence of nodes,* $\{g_0, ..., g_m\}$, *such that the output of* g_i *is an input of* g_{i+1}.

Definition 3.4 *Each node* g *in a combinational circuit has a* **weight** $w(g)$. *The value of node* g *at time* t *is that determined by a static evaluation of the node using the values on its inputs at* $t - w(g)$.

Definition 3.5 *We define* **delay** *as follows:*

1. *The delay through a path* $P = \{g_0, ..., g_m\}$ *is defined as* $d(P) = \sum_{i=0}^{m} w(g_i)$. *This is also called the* length *of the path.*
2. *The delay at a gate* $d(f) = w(f) + max\{d(i) | i \in inputs(f)\}$
3. *When delays in more than one network are under consideration, the notation* $d_N(f)$ *denotes the delay of node* f *in network* N.

We assume that the wires of a circuit act as ideal capacitors; that is, once assigned a value the wire holds that value until changed by a computation at its source node. We assume that the primary input vector c_2 has become available at $t = 0$; all inputs change simultaneously then. Further, for all negative values of t, the wires of the circuit hold the static values determined by some input vector c_1. Notationally, we capture this assumption by speaking of the value of function f at time t, $f(c_1, c_2, t)$, where c_1 is the input vector from $-\infty \leq t < 0$, and c_2 is the input vector from $0 \leq t \leq \infty$. Clearly $f(_, c_2, t) = f(c_2)$ for $t \geq d(f)$, since after time $t = d(f)$, f has assumed its static (final) value.

Definition 3.6 *An* **event** *is the transition of a node from a value of 0 to 1, or vice-versa.*

We envision a sequence of events $\{e_0, ..., e_m\}$, each e_i occurring at node f_i, and each event e_i occurring as a direct consequence of event e_{i-1}. We say that event e_0 *propagates* down path $\{f_0, ..., f_m\}$.

Definition 3.7 *A path $P = \{f_0, ..., f_m\}$, f_0 a primary input, is* **sensitizable** *if some event e_0 may propagate down this path to the output f_m.*

Definition 3.8 *The* **critical path** *of a network is its longest sensitizable path.*

This permits us to consider the boolean conditions for a path to be sensitizable. Let event e_i be the transition of node f_i from 0 to 1. Event e_{i+1} is the transition of f_{i+1} from either 0 to 1 or 1 to 0. In the former case, we have that f_{i+1} tracks f_i, in the latter, f_{i+1} tracks $\overline{f_i}$. The conditions under which this is possible is a boolean function, the arguments of which we call *side inputs.*

Definition 3.9 *Let $P = \{f_0, ..., f_m\}$ be a path. The inputs to f_i that are* **not** *f_{i-1} are called the* **side inputs** *to P at f_i. We denote the set of side inputs as $S(f_i, P)$.*

Using the Shannon expansion, we can write the condition for f_i to track f_{i-1} as $f_{i f_{i-1}} \overline{f_{i \overline{f_{i-1}}}} = 1$. Similarly, for f_i to track $\overline{f_{i-1}}$, we must have $\overline{f_{i f_{i-1}}} f_{i \overline{f_{i-1}}} = 1$. Since the sensitizing condition is that one of these must hold, we may write the sensitizing condition as:

$$\frac{\partial f_i}{\partial f_{i-1}} = f_{i f_{i-1}} \overline{f_{i \overline{f_{i-1}}}} + \overline{f_{i f_{i-1}}} f_{i \overline{f_{i-1}}} \quad (1)$$

or, equivalently, as

$$\frac{\partial f_i}{\partial f_{i-1}} = f_{i f_{i-1}} \oplus f_{i \overline{f_{i-1}}} \tag{2}$$

$\frac{\partial f_i}{\partial f_{i-1}}$ is described in the testing literature [1] [18], and is referred to there as the *boolean difference.*

Now, clearly we must have $\frac{\partial f_i}{\partial f_{i-1}} = 1$ when event e_{i-1} is propagated through f_i. We denote the time of event e_i, $t(e_i)$, as τ_i.

Lemma 3.1 *Let e_0 propagate down path $\{f_0, ..., f_m\}$, f_0 a primary input. Then $t(e_i) = \sum_{j=0}^{i} w(f_j)$.*

Theorem 3.2 *A path $\{f_0, ..., f_m\}$, f_0 a primary input, is sensitizable iff $\exists$ input vectors $c_1, c_2 \ni \forall i \ \frac{\partial f_i}{\partial f_{i-1}}(c_1, c_2, \tau_{i-1}) = 1$.*

Sketch of Proof: Follows easily from the definition of boolean difference and from the previous lemma, since the event can propagate through stage i iff the sensitizing condition holds at the time the event is generated, which is τ_{i-1}. ∎

4 Sensitization Conditions: Robustness and Families of Circuits

The delay model used in timing analysis methods is a worst case model; it is intended to provide an upper bound for the delay of all circuits which may be manufactured and operated in particular environments. A real circuit is not the idealized circuit of timing models; it is a circuit with the same topology, but with possibly smaller delays at some of the nodes. Hence the estimate provided by the algorithm must hold for an entire *family* of circuits, the "slowest" – in the sense of having the slowest components – of which is the one under analysis. Hence it is important that any critical delay algorithm be *robust* in the following sense: if the delays on some or all gates in the network are reduced, then the critical delay estimate provided by the algorithm is not increased. When the algorithm is applied to the worst-case circuit, a robust criterion thus guarantees that the estimate obtained is valid for any circuit in the family. Colloquially, we refer to this robustness property as the *monotone speedup* property.

Recall that every valid criterion must meet the monotone speedup property: if the delays on some or all gates in the network are reduced, then

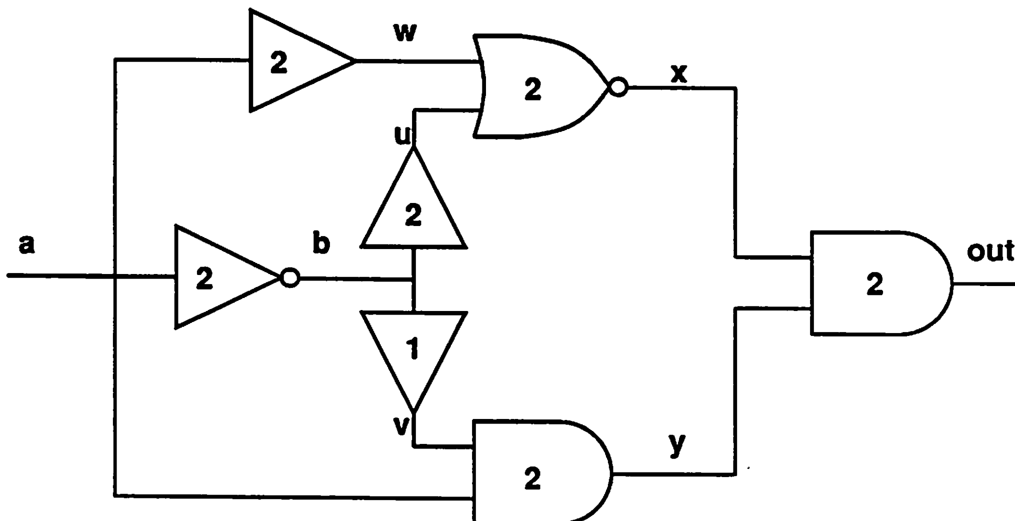


Figure 6: Monotone Speedup Failure

the critical delay estimate for the network is not increased. This guarantee cannot be given by the dynamic sensitization criterion, because the sensitizability of a path is inherently determined by the precise internal delays of the circuit. Hence, one can speed up a circuit and thus make a previously-unsensitizable path sensitizable. This path may be arbitrarily long (though not the longest in the circuit if such is unique); in particular, it may be longer than the longest-sensitizable path in the slower network.

Example: Consider the single-input circuit in figure 6. Assume the delay on all gates are as marked. Note when a is toggled from 1 to 0 at $t = 0$, from $t = 2$ to $t = 3$ there is a 0 on both u and w, so $x = 1$ from $t = 4$ to $t = 5$. However, in this case $y = 0$ throughout, so $out = 0$ throughout. Similarly, when a toggles from 0 to 1, from $t = 0$ to $t = 2$ there is a 1 on each input of y, so $y = 1$ from $t = 2$ to $t = 4$. However, in this case $x = 0$ throughout, so $out = 0$ throughout. This circuit therefore has no dynamically sensitizable paths and its delay is 0.

If we now speed the circuit up by removing the delay buffer between b and u, so that u now arrives at $t = 1$, but all other delays are unchanged, when a is toggled from 0 to 1 we have a zero on each input to x from $t = 1$ (when u turns from 1 to 0) to $t = 2$ (when w turns from 0 to 1). Hence $x = 1$ from $t = 3$ to $t = 4$. But $y = 1$ from $t = 2$ to $t = 4$, so $out = 1$

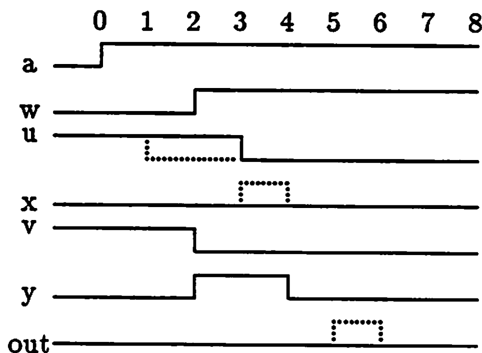


Figure 7: Timing Diagram of Monotone Speedup Failure

from $t = 5$ to $t = 6$. Hence there is at least one dynamically sensitizable path in this circuit of length 6; by reducing the delay on the wire from the inverter to u from 2 to 0, we have increased the critical delay on this circuit from 0 to 6. A full timing diagram of the situation appears in figure 7. In this diagram, the solid lines represent the behaviour of the "slow" original circuit; the dotted lines the behaviour in the sped-up, or "fast" circuit.

This phenomenon – that one can demonstrate circuits where the longest true path of a circuit increases length as components are sped up – appears to hold in every level-sensitive logic where each wire holds its value until the value is changed. In fact, given that the exact delay times at the nodes in a circuit are only determined up to some given tolerance, the sensitizability of a path within a given circuit may vary between two "identical" but separate realizations. *Hence the longest sensitizable path appears to be an inherently nondeterminate property of logic circuits.*

5 The Viability Theory

A stronger property that satisfies these constraints is *viability*. Before we formally introduce the concept of viability, we wish to introduce its motivation.

Fundamentally, a node f_i is dynamically sensitized to an input f_{i-1} at τ_{i-1} but not statically sensitized to f_{i-1} only if the value of the function $\frac{\partial f_i}{\partial f_{i-1}}$ changes value at τ_{i-1} or later. This can only occur if there are events

on some set of inputs to $\frac{\partial f_i}{\partial f_{i-1}}$ at or after τ_{i-1}; these are called *late side inputs.* Under these conditions, we may assume that each of these inputs are at *any* value at τ_{i-1}, and hence (to be conservative), we assume that they are set to any value which will propagate the event. Mathematically, we do this by "smoothing" the function $\frac{\partial f_i}{\partial f_{i-1}}$ over the late inputs.

Definition 5.1 *For any function f and variable x, the* **smoothing operator** *$S_x f$ is defined as:*

$$S_x f = f_x + f_{\overline{x}}$$

Intuitively, we ignore x when calculating the value of $S_x f$. It is easy to prove that $S_x S_y f = S_y S_x f$. We may thus define, for a set $U = \{x_1, .., x_n\}$, $S_U f = S_{x_1} S_{x_2} ... S_{x_n} f$, and for $U = \emptyset$, $S_\emptyset(f) = f$.

Definition 5.2 *Consider a path $P = \{f_0, ..., f_m\}$. Q is said to be a* **side path** *of P at f_i if Q terminates in g, a side input to P at f_i.*

Definition 5.3 *A path $P = \{f_0, ..., f_m\}$ is said to be* **viable** *under an input cube c if, at each node f_i there exists a (possibly empty) set of side inputs $U = \{g_1, ..., g_n\}$ to P at f_i, such that, for each j,*

1. *g_j is the terminus of a path Q_j,*
2. *$d(Q_j) \geq \tau_{i-1}$ and Q_j is viable under c*
3. *$(S_U \frac{\partial f_i}{\partial f_{i-1}})(c) = 1$*

Intuitively, at each node we find the conditions which simultaneously permit a set of side inputs U (a subset of the *late side inputs*) to undergo events later than τ_{i-1}, and the remaining side inputs to statically sensitize the node. It is important to note that this can only occur if there is some assignment to the variables in U which statically sensitizes the node; the effect of the smoothing operator is to permit this assignment to be made, independent of conditions elsewhere in the network. Effectively, the variables in U are made independent variables by the smoothing operator. Note that the case $U = \emptyset$ corresponds to static sensitization.

In [15], it is shown that the viability criterion is both correct (in that all dynamically sensitizable paths are viable) and robust (in that the longest viable path in a "fast" network is no longer than the longest viable path in a "slow" network). In preparation is a report that demonstrates that viability is the tightest robust, correct criterion.

5.1 The Viability Function

At some very fundamental level, any set condition may be expressed as a multiple input logic function. We are interested in making explicit the logic function that underlies viability, because the computation of the viable paths may be made more efficient through explicit computation of this function, because various properties may be proved through use of this function, and because we shall develop a powerful theorem which permits us to quickly establish the correctness of approximation procedures.

The viability function ψ_P on a path P is easy to define: $\psi_P(c) = 1$ iff P is viable under c. This hardly permits us more insight than the viable path definition, however. Now, we prefer to define ψ_P in terms of some function $\psi_P^{f_i}$ at each node f_i. We develop this function intuitively, then justify its definition in a theorem at the end of this section.

Intuitively, we expect that the function ψ_P will be the product of the viability conditions at each node f_i, which in turn are captured in the function $\psi_P^{f_i}$

$$\psi_P = \prod_{i=0}^{m} \psi_P^{f_i}$$

Since time plays an important part in the definition of viable path, it is convenient to capture it in the definition of the viability function. For any node g, let $\mathcal{P}_{g,t}$ be the set of paths terminating in g of length at least t. Then

$$\psi^{g,t} = \sum_{Q \in \mathcal{P}_{g,t}} \psi_Q$$

Letting U be any subset of $S(f_i, P)$ we can express the viability condition on the subset as:

$$(S_U \frac{\partial f_i}{\partial f_{i-1}}) \prod_{g \in U} \psi^{g,\tau_i - 1}$$

since this condition must be satisfied for one subset, we may write:

$$\psi_P^{f_i} = \sum_{U \subseteq S(f_i,P)} (S_U \frac{\partial f_i}{\partial f_{i-1}}) \prod_{g \in U} \psi^{g,\tau_i - 1}$$

In summary, we define:

Definition 5.4 *The set of paths which terminate in g of length $\geq t$ are denoted $\mathcal{P}_{g,t}$.*

Definition 5.5 *The* viability function *(also* viable set*) of a path $P = \{f_0, ..., f_m\}$ is defined as:*

$$\psi_P = \prod_{i=0}^{m} \psi_P^{f_i} \tag{3}$$

where

$$\psi_P^{f_i} = \sum_{U \subseteq S(f_i, P)} (S_U \frac{\partial f_i}{\partial f_{i-1}}) \prod_{g \in U} \psi^{g, \tau_i - 1} \tag{4}$$

and

$$\psi^{g,t} = \sum_{Q \in \mathcal{P}_{g,t}} \psi_Q \tag{5}$$

We have immediately:

Theorem 5.1 *$P = \{f_0, ..., f_m\}$ is viable under some minterm c iff c satisfies ψ_P.*

6 τ-Irredundant Faults

We now have the mathematical and conceptual tools in place to consider the concept of *τ-irredundant faults.* Broadly, we want a definition which will permit us to determine when a stuck fault will cause a circuit to report an incorrect value when the output is sampled at time τ. Note that our previous discussion of the uncertainty in delays requires that we consider the entire family of circuits represented by a single circuit, N. We often refer to such a family as the network family N.

Notation: Each multiple-fault c on a network family N denotes a family of faulty networks. This faulty family will be denoted N_c. To each network N' in the network family N there exists a corresponding faulty network N_c' in N_c. If a node in N is denoted f, its counterpart in N_c is denoted f_c.

Definition 6.1 *A single- or multiple-stuck-at-fault c on a network family N, inducing faulty family N_c, is said to be τ-***redundant** *iff, for each output f of N, $f_c \oplus f \equiv 0$ at all $t \geq \tau$ in* every *network N' in the family represented by N. If a fault is not τ-redundant then it is τ-***irredundant.** *If a circuit is τ-irredundant, then there is some input vector v, which excites the irredundancy condition and we say that the fault is* **tested** *by v.*

In sum, the presence of the irredundant multiple stuck-fault c causes some network in the family to misbehave at some time $t \geq \tau$.

Now, it is clear that conventional (time-independent) multiple stuck-faults fit nicely into this picture as a special case, namely ∞-irredundant stuck faults. This gives us a hint that we are looking at a *spectrum* of irredundant (or redundant) faults, parameterized in the time axis. The next theorem makes this clear.

Theorem 6.1 *Let N be any network family. Let $\tau_0 \geq \tau$, where τ is the maximal length of the critical path over the family N. c is a τ_0-irredundant multiple-stuck fault in N, and is tested by input vector v, iff:*

1. *c is a conventionally irredundant multiple-stuck fault in N, and is tested by input vector v; or*
2. *there is no dynamically-sensitizable path, terminating in an output, of length $> \tau_0$ in any network in the family represented by N, but there is such a path in the family represented by the faulty network N_c, and, further, there exists an input vector v_1 such that v_1 and v exercise the path in N_c.*

Proof: For the first item, clearly any conventionally-irredundant fault is also τ_0-irredundant (set $t = \infty$). For the second, assume there is such a path. Then in some network in the family N_c, when v_1 and v are applied in succession, there is an event on output f_c at some time $t > \tau_0$. Choose $\epsilon < t - \tau_0$. Now, the value of f_c at $t - \epsilon$ differs from the value of f_c at $t + \epsilon$. The value of f in N is fixed to its final value at some time $t_0 < \tau_0$, and hence must disagree with the value of f_c in the faulty family N_c at either $t + \epsilon$ or $t - \epsilon$. Both these latter times are $> \tau_0$, and hence the fault is τ_0-irredundant. Conversely, assume the fault is τ_0-irredundant. Then by definition the faulty family N_c disagrees with the good family at some time $t > \tau_0$. If some output changes value at t, then clearly there is a dynamically sensitizable path in the faulty family exercised by v_1 and v, and so the second item is satisfied since there is no dynamically-sensitizable path in the good family of length $> \tau_0$. Otherwise, the value of the faulty family at ∞ equals its value at τ_0, and must differ from the value of the good family at τ_0 (which is also its value at ∞). If these values agreed, there is no fault, and hence they must disagree in the values at ∞. In this case, c is a conventionally-irredundant stuck fault. ∎

Corollary 6.2 *Let c be a τ_1-irredundant multiple stuck-fault in a network family N with maximal critical path length τ. Then c is also τ_0-irredundant for all $\tau_1 \geq \tau_0 \geq \tau$.*

The clearly interesting set of faults is the τ_k-irredundant set, where τ_k is the length of the clock; clearly $\tau_k \geq \tau$, where τ is the maximal critical-path length over the network-family N. We wish to characterize τ_k-irredundant multiple stuck faults. In order to do this, we need a conjecture, which seems quite likely to be true.

Conjecture 6.1 *Every path in a network-family N, viable under c, is dynamically sensitizable in some member of N.*

If this conjecture is in fact true, then the identity of dynamically sensitizable paths over a family with the set of viable paths in the slow member of the family gives us a tight criterion for finding such irredundant faults, and a correct (though expensive) method of identifying these faults: set the candidate fault, run a timing simulator using the viability criterion (e.g., [14]) over the faulty network, and determine if there are any viable paths of length $> \tau_k$. Note that this procedure will return a superset of the τ_k-irredundant multiple stuck-faults if the identity above is inexact, since it is known that the set of viable paths of a network contain a path at least as long as the longest dynamically sensitizable path over a network family. Further, any critical-path-correct sensitization criterion [15] used in conjunction with this procedure will return such a superset. Further, the test vectors associated with the multifault is the satisfying set of the sensitizing function for the path in the faulty network. Hence this procedure not only verifies a τ-irredundant multifault, it also gives the test vector for it.

This procedure is expensive and inelegant, however. It is on a par with setting a conventional multiple stuck-fault in a network N and then computing the satisfying set of $N_c \oplus N$ to obtain a test vector for the fault. We wish to characterize the set of such multifaults in a way that permits us to localize the search for these faults in the graph.

Note that ∞-irredundant multifaults may be found by conventional test generation means. What we are interested in is the set of multifaults which are τ_k-irredundant but not ∞-irredundant.

Intuitively, we expect that the set of τ_k-irredundant multifaults to set static sensitizing values on side inputs to a long false (in the original network family) path. Something quite like this is in fact the case.

Theorem 6.3 *Let c be a τ_k-irredundant multifault which is not ∞-irredundant, and which, when set, makes viable a path $P = \{f_0, ..., f_m\}$ with $d(P) > \tau_k$. Then c is fault-equivalent to a τ_k-irredundant multifault c' with the following properties:*

1. *The edges directly set by the fault c' are all side inputs to P.*
2. *The edges set by c' are all set to sensitizing values; i.e., if edge g is set by c', $g \in S(f_i, P)$, then g is set to a value satisfying $\frac{\partial f_i}{\partial f_{i-1}}$.*

This proof really needs work. But I'm more interested in giving you the outline of the proof here

Proof: Since c makes viable a path $P = \{f_0, ..., f_m\}$ with $d(P) > \tau_k$, then for some f_i on P $\psi_P^{f_i}$ is increased (some may be decreased, but these can be neglected). Using the expansion for $\psi_P^{f_i}$, we have that:

$$\psi_P^{f_i} = \sum_{U \subseteq S(f_i, P)} (S_U \frac{\partial f_i}{\partial f_{i-1}}) \prod_{g \in U} \psi^{g, \tau_i - 1}$$

is increased. Hence for some set $U \subseteq S(f_i, P)$ we have that $(S_U \frac{\partial f_i}{\partial f_{i-1}}) \prod_{g \in U} \psi^{g, \tau_i - 1}$ is increased. We have two cases:

1. $(S_U \frac{\partial f_i}{\partial f_{i-1}})$ is increased. In this case, we must have that some $h \in S(f_i, P) - U$ is set to a sensitizing value by c; or
2. $\psi^{g, \tau_i - 1}$ is increased for some $g \in U$. But this effect can be achieved by the simple device of setting each such g to its sensitizing value in the multifault. Once this is done, any settings in the multifault which served only to increase $\psi^{g, \tau_i - 1}$ can be replaced by these settings.

c' is obtained from c by replacing all the settings which serve to increase $\psi^{g, \tau_i - 1}$ with the relevant g set to sensitizing values. ■
[2]

Now, unfortunately this does not translate well to single stuck-faults; it does not appear to be the case that every τ_k-irredundant single-stuck fault is fault-equivalent to a τ_k-irredundant single-stuck fault on a side input to the long path made viable by the fault. This is due to two effects, which may occur either separately or jointly.

[2] The use of the phrase "sensitizing value" is a little misleading here. For most gates, f_i, $\frac{\partial f_i}{\partial f_{i-1}}$ is a single cube, and hence only one value of g will sensitize the cube. For other gates, any satisfying assignment may be chosen

1. A single stuck-fault potentially sets many nodes to values in the network through direct simulation, i.e., it causes a multifault, which in turn may directly sensitize the long path. This can be traced using direct forward simulation of the fault, and the relevant multifault extracted.

2. A single stuck fault may occur as a side input to a side path, i.e., may make a long side path to the long path viable, which in turn makes the long path viable. In this case the relevant single stuck-fault is fault-equivalent to the appropriate single-stuck fault at the head of the long side path.

Each of these separate effects are easy to investigate. However, the combination of these effects are problematic. If it is believed that an interaction of these two effects giving rise to a τ_k-irredundant multifault is unlikely, then simply processing the single stuck-faults on the side inputs to the long paths is sufficient to find most or all of the τ_k-irredundant single stuck faults. Experimental work on this needs to be done.

Note that this theorem implies that the polarity of the relevant multifault is quite firmly fixed: each side input involved in the multifault must be stuck at a *sensitizing* value. This information, combined with the well-known information that almost all single-stuck faults are ∞-irredundant, leads to the conclusion that relatively few single-stuck faults need be tested, and hence that the relatively expensive procedure of setting the fault and then doing a timing simulation on the faulty network in order to obtain the test set is acceptable.

This section really needs to be beefed up

7 Observability of τ-Irredundant Faults

It is insufficient to simply generate tests for τ-irredundant faults. The effect of setting the single- or multifault must be observed on the fabricated wafer. Sadly, timing analysis literature has little encouraging to say on this problem, even if the network can be tested at speed.

The difficulty lies in the concept of the network family. The viability theory, even if the conjecture is true, does not guarantee that the path found to be viable will be dynamically sensitizable in the network under test; rather, (at best) it only guarantees that the path will be dynamically sensitizable in *some* network in the family. There is no guarantee that the die under test on the fabrication line is that circuit. Worse, even a

single fabricated circuit is not a single circuit throughout its lifetime; since it operates in a variety of thermal and electrical environments, and so gives rise to a circuit family, of which the circuit under test is not necessarily the slowest. Finally, such networks generally undergo hazards; sampling at τ_k does not guarantee that two events at some primary output do not lie off in the future: merely guaranteeing that the output at τ_k is equal to the good output is no guarantee that a τ_k-irredundant fault does not exist; it merely guarantees that an even number exist. Further, if one of the affected paths speeds up during operation, at some point the chip may latch in the value during the hazard. *Hence, if one fails to observe the fault on the process line, that is no guarantee that the fault does not exist or that, if it does exist, it is irrelevant. The fault may exist, and, during the lifetime of the chip, cause the circuit to malfunction.* This point cannot be stressed too much. New techniques for observing the fault must be devised, at least in general logic networks (the problem is somewhat alleviated in precharged-unate networks such as DCVS). Two possibilities include:

1. Built-in Self Test (BIST) techniques of a new form. Standard BIST uses a set of feedback-shift registers to compute a signature of the network, and checks this against an ideal signature. This consumes little area, and finds almost all irredundant stuck faults. Whether standard BIST techniques will find all τ-irredundant stuck faults with the same high probability is an open question. However, it is clear that if only a very few test vectors need be stored to exercise the τ-irredundant faults, then these (and the good and faulty solutions) may be stored in a small on-chip ROM. Either approach has the great advantage of testing the circuit at speed in the operating environment. Ideally, such checks should be done relatively frequently;

2. Direct observation of critical edges, e.g., those involved in a τ_k-irredundant multifault but not in an ∞-irredundant multifault. Since the redundant multifault is controllable, direct observation removes this difficulty at some cost in circuit area, and permits standard fabrication-line testing.

One other possibility is to set the clock to tolerate a single stuck failure on a τ_k-irredundant edge. For example, a single stuck failure on a carry-bypass line will not increase the delay by a factor of 2, to the delay of the full ripple-carry circuit, but, rather, only by about 4 gate delays; in other words, in the full 32-bit bypass adder with block size 4, there are eight

τ-irredundant single stuck faults but no $(\tau + 4)$-irredundant single stuck faults. There is obviously at least one $(\tau + 4)$-irredundant stuck multifault. The safe clock rate τ_k can be determined by setting each suspected single fault, running the appropriate timing verifier, and finding the clock rate, and repeating for all single stuck faults. The acceptability of this technique is likely to depend upon the design environment, and the extent to which τ_k exceeds τ.

8 τ-Irredundant Faults on Precharged-Unate Networks

In [16] it is shown that the dynamic sensitization criterion is robust on precharged-unate networks such as DOMINO[11], NORA[7], and DCVS[8]; a set of equations used to define the dynamic sensitization criterion on such networks were defined, similar to the viability equations. Further, such networks are known to be hazard-free.

An analogue to theorem 6.3 holds for such networks, so, again, only multifaults incident on a long false path need be considered.

Such networks are extremely well-behaved from a timing point of view. If electrical conditions on the fabrication line are set to worst-case, then it can be guaranteed that if $f_c \oplus f = 0$ when a test vector for a τ-irredundant fault is applied to the appropriate die under test, then either the fault has not occurred on the chip or it will be irrelevant throughout the lifetime of the circuit under normal operating conditions.

References

[1] S. B. Akers. On a Theory of Boolean Functions. *J. SIAM*, 1959.

[2] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. Multi-level logic minimization using implicit don't cares. *IEEE Transactions on CAD*, 1988.

[3] J. Benkoski, E. Vanden Meesch, L. Claesen, and H. DeMan. Efficient algorithms for solving the false path problem in timing verification. In *IEEE International Conference on Computer-Aided Design*, 1987.

[4] Daniel Brand and Vijay S. Iyengar. *Timing Analysis Using Functional Analysis.* Technical Report RC 11768, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 10598, 1986.

[5] M. A. Breuer. The effects of races, delays, and delay faults on test generation. *IEEE Transactions on Computers*, 1974.

[6] C. Thomas Glover and M. Ray Mercer. A method of delay fault test generation. In *Design Automation Conference*, 1988.

[7] Nelson F. Goncalves and Hugo J. DeMan. NORA:a racefree dunamic CMOS technique for pipelined logic structures. *IEEE Journal of Solid State Circuits*, 1983.

[8] L. G. Heller, W. R. Griffin, J. W. Davis, and N. G. Thoma. Cascode voltage switch logic: a differential CMOS logic family. In *IEEE International Solid State Circuits Conference*, 1984.

[9] V. M. Hrapcenko. Depth and delay in a network. *Soviet Math. Dokl.*, 1978.

[10] N. Jouppi. TV: an nMOS timing analyzer. In *Third Caltech VLSI Conference*, 1983.

[11] R. H. Krambeck, C. M. Lee, and H-F. S. Law. High-speed compact circuits with CMOS. *IEEE Journal of Solid State Circuits*, 1982.

[12] Jean Davies Lesser and John J. Shedletsky. An experimental delay fault test generator for LSI logic. *IEEE Transactions on Computers*, 1980.

[13] Chin Jen Lin and Sudhakar M. Reddy. On delay fault testing in logic circuits. *IEEE Transactions on CAD*, 1987.

[14] Patrick C. McGeer and Robert K. Brayton. Efficient algorithms for computing the longest viable path in a combinational network. In *Design Automation Conference*, 1989.

[15] Patrick C. McGeer and Robert K. Brayton. Provably correct critical paths. In *Design Automation Conference*, 1989.

[16] Patrick C. McGeer and Robert K. Brayton. Timing verification on precharged-unate networks. *In Preparation*, 1989.

[17] John K. Ousterhout. Crystal: a Timing Analyzer for nMOS VLSI Circuits. In *Third Caltech VLSI Conference*, 1983.

[18] Frederick F. Sellers, Jr., M. Y. Hsiao, and L. W. Bearnson. Analyzing errors with the Boolean difference. *IEEE Transactions on Computers*, 1968.