Universal Delay Test Sets for Logic Networks

Uwe Sparmann, *Member, IEEE*, Holger Müller, and Sudhakar M. Reddy, *Fellow, IEEE*

Abstract— **It has been shown earlier that, if we restrict to unate gate network (UGN) realizations, there exist universal test sets for boolean functions. Such a test set only depends on the function f, and checks any UGN realization of f for all multiple stuck-at faults and all robustly testable stuck-open faults. In this paper we prove that these universal test sets are much more powerful than implied by the above results: They also constitute complete delay fault test sets for arbitrary UGN implementations of a given function. This is even true for UGN networks which are not testable with respect to the gate or path delay fault model. Our ability to prove the temporal correctness of such circuit realizations, comes from the fact that we do not argue the correctness of individual paths, but complete path systems.**

Keywords— **Delay test, unate gate networks, universal test sets, design for testability.**

I. Introduction

THE development of VLSI systems is driven by the demand for higher scale of integration and faster circuits. As a consequence, the susceptibility to manufacturing defects rises, and reliability becomes a serious problem. In order to achieve a sufficient quality level of the shipped product, chips have to be checked not only to verify their static correctness, but also to guarantee their correct dynamic (temporal) behavior.

Test generation methods for checking the temporal correctness of a manufactured circuit have been studied considerably in the literature (see for example [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]). They are mostly based on the gate [13] or path delay [14] fault model. Recently, the path delay fault model has been generalized, by also considering path systems in case that the speed of single paths cannot be checked robustly [15]. This approach has been successfully applied to derive less stringent necessary and sufficient conditions for the delay testability of two-level circuits.

The above test generation methods assume that the realization of the circuit is known *a priori*. Thus, a test set with near minimal size can be computed. At the same time, since the test set is tailored to a specific design, even the smallest change in circuit structure may render it useless. Also, the test generation process for a design dependent delay test set is computationally very expensive. (In order to achieve complete fault coverage with respect to the path delay fault model, the number of paths which have to be considered may be exponential in the number of circuit inputs.) In addition, full testability with respect to the gate or path delay fault model is usually only achievable by extensive design for testability modifications or synthesis for testability [16], [17], [18], [19], [20], [21], [22], [23], [24], which often increase circuit size and can degrade circuit performance.

The purpose of this work is to show that for a large class of logic circuits, the *unate gate networks* or UGNs [25], the above problems can be eliminated, i.e. there exist easy to compute *universal delay test sets* which only depend on the desired function but not its specific UGN implementation. Previous results concerning the derivation of universal test sets for UGNs [26], [27], [25], [28], [29] have been restricted to the multiple stuck-at and the stuck-open fault model. Betancourt, Akers, and Reddy (Gupta and Jha) have shown how to derive a universal test set for a given function f, which checks any UGN realization of f for multiple stuck-at faults [26], [27], [25] (all robustly testable stuck-open faults [28]). In this paper, we prove that these universal test sets are much more powerful than implied by the above results. They also constitute complete delay fault test sets for arbitrary UGN implementations of the given function. This result implies that for UGNs complete testability for delay faults can be guaranteed without any design for testability modifications. This is even true for UGN realizations which are not testable with respect to the gate or path delay fault model. In order to be able to prove the temporal correctness of such implementations, we pursue the approach of [15] and argue the correctness of whole path systems instead of individual paths. The notions and techniques developed in this context are not only of interest in the context of universal test sets but also for more general settings [30].

There are various applications of our results. The first one is for testing of dynamic CMOS logic [31], [32], [33], which is often applied in high speed circuits [34] and must be structured as a unate gate network. Also self-checking circuits based on unordered codes, are usually built as UGNs [35], [36]. Another application is in synthesis for delay fault testability. Any circuit can be transformed into a UGN of the same depth with at most doubling its size [25]. This implies a design for delay-testability method which does not compromise the speed of a given circuit implementation. Finally, it will be proven in this paper that for delay testing of UGNs only two initialization vec-

This work has been supported by DFG, Grant No. Sp431/1-1 and SFB 124 'VLSI Entwurfsmethoden und Parallelität'.

U. Sparmann and H. Müller are with the Computer Science Department, University of Saarland, 66123 Saarbrücken, Germany.

S.M. Reddy is with the Department of ECE, University of Iowa, Iowa City, IA 52242, USA.

tors are sufficient. This fact can be applied in order to simplify self test of UGNs for delay faults. Experimental results given for ISCAS89 [37] and two-level MCNC [38] benchmarks show that for many functions there exist efficient UGN implementations, and that the corresponding universal test sets are of reasonable size.

The paper is organized as follows: In Section II we recall some basic definitions concerning UGN realizations of boolean functions. After a short review of previous work in the field of universal test sets, a formal statement of our new results is given. Section III gives the corresponding proofs. For simplicity of presentation the statement of the results in Section II and the proofs in Section III are restricted to unate functions. Section IV shows how to generalize our results to arbitrary functions. Experimental results illustrating the practical applicability of our methodology are given in Section V.

II. Basic Definitions and Main Results

We will start this section by recalling some basic facts about unate gate network realizations of boolean functions [25]. After that we briefly review the results of [26], [27], [25], [28] concerning the testability of unate functions by universal test sets, and give our new results. For simplicity of presentation we will restrict to unate functions in the following. How to extend our results to arbitrary functions will be discussed in Section IV.

A. Preliminaries

We will restrict to single output circuits in the following, for multiple output circuits the theory can be applied for each output cone separately. First we review some well known notions from the theory of unate functions (see for example [39]).

Definition 1: Let $f : \mathbf{B}^n \to \mathbf{B}$ ($\mathbf{B} = \{0, 1\}$) be a function depending on all of its input variables.

- f is *positive (negative) unate in variable* $x_i \iff$ for every input vector $v \in \mathbf{B}^n$ changing the value of x_i from 0 to 1 (1 to 0) never changes the function value from 1 to 0. Otherwise, f is said to be *binate* in x_i.
- f is called *unate* iff it is unate in all of its input variables.

For a given unate function f we associate a *parity* $\pi_i \in \mathbf{B}$ with each of its input variables such that:

$\pi_i = 0$ ($\pi_i = 1$) $\iff$ f is positive (negative) unate in input variable x_i.

For $\alpha, \pi_i \in \mathbf{B}$ let:

$$\alpha^{\pi_i} := \begin{cases} \alpha & \text{if } \pi_i = 0 \\ \overline{\alpha} & \text{if } \pi_i = 1 \end{cases}$$

i.e. π_i gives the 'number of inversions' of variable α. (Note, that the above definition is different from the usual definition of powers where $\alpha^1 = \alpha$.)

Definition 2: A partial order $>_f$ for the input vectors of a unate function f is defined as follows:

$$\begin{aligned}(v_1, \ldots, v_n) >_f (w_1, \ldots, w_n) \iff \\ (\forall i \in \{1, \ldots, n\} : v_i^{\pi_i} \geq w_i^{\pi_i}) \\ \wedge\ (\exists i \in \{1, \ldots, n\} : v_i^{\pi_i} > w_i^{\pi_i})\end{aligned}$$

Comparing $>_f$ to the usual order $>$ on $\mathbf{B}^n$, it holds that π_i ($\overline{\pi_i}$) takes over the part of 0 (1). More exactly:

Observation 1: Input vector $min_f := (\pi_1, \ldots, \pi_n)$ ($max_f := (\overline{\pi_1}, \ldots, \overline{\pi_n})$) is the minimum (maximum) element of $\mathbf{B}^n$ with respect to the partial order $>_f$.

Definition 3: An input vector v for function f is called a *true (false) vertex* of f if and only if $f(v) = 1$ ($f(v) = 0$).

- The set of all true (false) vertices of function f will be denoted by $TRUE_f$ ($FALSE_f$).
- $v \in TRUE_f$ ($v \in FALSE_f$) is called a *minimal true vertex (maximal false vertex)* if there exists no true (false) vertex w such that $v >_f w$ ($w >_f v$).
- The set of all minimal true (maximal false) vertices of function f will be denoted by $MINT_f$ ($MAXF_f$).

Example 1: Consider the function $f = x_1x_2x_3 \vee x_2x_3x_4 \vee x_3x_4x_5$. Since f is positive unate in all its input variables, we have: $\pi_i = 0\ \forall i \in [1 : 5]$. The minimum (maximum) element of the partial order $>_f$ is $min_f = (0\ldots0)$ (resp. $max_f = (1\ldots1)$). The set of minimal true (maximal false) vertices is given by $MINT_f = \{(11100), (01110), (00111)\}$ ($MAXF_f = \{(11011), (01101), (10101), (10110)\}$). (Assignments to the input variables are specified in the order $x_1, \ldots, x_5$.)

Next we introduce the circuit model used in this paper: A circuit consists of primary inputs (PIs), primary outputs (POs), gates, and fan-out points. We will restrict to AND, NAND, OR, NOR, and NOT gates in the following. The PIs, POs, and gates are connected to each other and the fan-out points by *lines*. The following definition is adapted from [25]:

Definition 4: Let F be a single output circuit as described above, and assume that F depends on all of its inputs.

- Consider two arbitrary lines l_1 and l_2 in F. For a path p from l_1 to l_2 the *inversion parity* $IP(p)$ *of* p is defined as the number of inverting gates (NAND, NOR, NOT) on p modulo 2.
- F is called a *unate gate network (UGN)* iff for any pair (l_1, l_2) of lines in the circuit it holds that all paths from l_1 to l_2 have the same inversion parity.

From the definition of unate gate network it follows directly that the function f computed by a UGN F is always unate. The relation between the input/output inversion parities of F and the parities of the variables of f is given by the following simple observation.

Observation 2: Consider a UGN F realizing function f. Let x_i be the input variable of f corresponding to primary input i of F, and let π_i denote the parity of x_i.

For any path p from PI i to the PO of F it holds that $IP(p) = \pi_i$.

Example 2: Figures 1 and 2 show two UGN realizations F_1 and F_2 of the function f given in Example 1. Since f is positive unate in all of its input variables, i.e. $\pi_i = 0\,\forall i \in [1:5]$, all paths from primary inputs to the primary output have inversion parity 0, i.e. contain an even number of inverting gates.

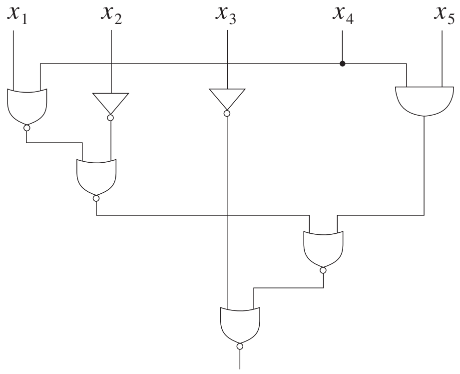

Fig. 1. Unate gate network F_1

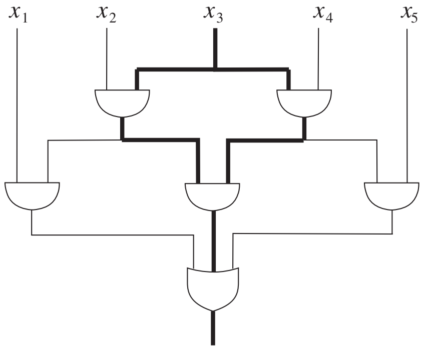

Fig. 2. Unate gate network F_2

B. Results

Betancourt, Akers, and Reddy have shown in [26], [27], [25] that, if we restrict to unate gate network realizations, there exist *universal test sets* with respect to the multiple stuck-at fault model which only depend on the considered function but not on the network structure.

Theorem 1 ([26], [27], [25]) Let f be a unate function and F an arbitrary UGN realizing f. F can be tested for all non-redundant multiple stuck-at faults with input vector set $T_f := MINT_f \cup MAXF_f$.

In [28] Gupta and Jha have generalized this result for the detection of robust testable stuck-open faults in unate gate networks. Let TT_f denote the following set of two pattern transition tests:

$$TT_f \;:=\; \{< min_f, t > | t \in MINT_f\} \cup \{< max_f, t > | t \in MAXF_f\}$$

(Remember that $min_f := (\pi_1, \dots, \pi_n)$ ($max_f := (\overline{\pi_1}, \dots, \overline{\pi_n})$) denotes the minimum (maximum) element of $\mathbf{B}^n$ with respect to the partial order $>_f$.)

A two pattern test for a stuck-open fault is called *robust*, if it cannot be invalidated by circuit delays or skews in the circuit input change [40].

Theorem 2 ([28]) Let f be a unate function and F an arbitrary UGN realizing f. TT_f contains a robust test for every stuck-open fault in F which is robustly testable.

In this paper, we will show that the test set TT_f is much more powerful than indicated by Theorems 1 and 2. It not only checks any UGN realization of f for stuck-at and stuck-open defects but also for delay faults.

For the exact statement of this result, we introduce our **delay fault model**, which has been adapted from [41]:
A *connection* is a circuit line or a (input pin)-to-(output pin) path through a gate. With each connection c we will associate a finite *rising (falling) delay*[1]. This delay is denoted by $\Delta_{0\to1}(c)$ ($\Delta_{1\to0}(c)$), and specifies the time necessary for the corresponding transition to propagate over connection c.
Due to manufacturing variations or defects, the delays of connections will vary for different implementations of the same circuit design. Thus, we will distinguish between the circuit design F and a (manufactured) implementation $I := (F, \Delta)$ of F, where Δ is a mapping assigning a rising and falling delay to each connection.

Definition 5: An implementation $I = (F, \Delta)$ (statically) realizing function f contains a *delay fault* for clock period τ, $\Longleftrightarrow$ there exists an input vector $v \in \mathbf{B}^n$ and an assignment $\mathcal{A}$ of values to the lines of F such that:

if v is applied at time 0 to I with initial assignment $\mathcal{A}$, then, the output of I at time τ does not equal $f(v)$.

The main result of this paper can now be stated as follows:

Theorem 3: Let f be a unate function and $I = (F, \Delta)$ an arbitrary UGN implementation of f. Assume that I operates correctly with respect to clock period τ for all tests from TT_f.[2] Then, I is free of delay faults for all clock periods $\geq \tau$.

[1] Infinite delays are not considered, since they correspond to stuck-at or stuck-open faults which are covered by Theorems 1 and 2.

[2] We assume the usual test application procedure for delay testing with two pattern tests $< t_1, t_2 >$, i.e.: (1) The circuit is allowed to stabilize on the initialization vector t_1 by applying t_1 'long enough', and (2) t_2 is applied, and the response of the circuit is checked after time τ.

The main argument against the application of universal tests (T_f and TT_f) with respect to the stuck-at and stuck-open fault model has been their size. This is due to the fact that the cardinality of (single) stuck-at and stuck-open fault test sets is at most proportional to the number of lines in the circuit, while the size of T_f and TT_f usually grows faster and might even be exponential in the number of PIs. The application of TT_f for verifying the temporal correctness of a circuit is more attractive, since delay fault testing techniques are usually more expensive in terms of the number of test patterns. (Not only gates or lines but paths have to be verified, and the number of paths can be exponential in the circuit size.)

Another point to note is that for most circuit realizations, techniques for completely verifying their temporal correctness do not exist. Testability for delay faults can only be guaranteed by design for testability modifications. From Theorem 3 it follows that such modifications are not necessary for UGNs.

Example 3: For the function f given in Example 1 $|MINT_f \cup MAXF_f| = 7$. Thus, test TT_f consists of only seven two pattern sequences, which check any UGN implementation of f (like for example arbitrary implementations of the circuits F_1 and F_2, see Figures 1 and 2) for stuck-at, stuck-open, and delay faults as given by Theorems 1, 2, and 3.

Note, that for circuit F_2 its temporal correctness cannot be verified based on the path delay fault model. This is due to the fact that the two paths drawn in bold lines in Figure 2, starting from input variable x_3 and going to the output of F_2, are not robustly [1] testable with respect to the $1 \to 0$ transition at their input. These paths are neither functionally redundant [8] nor both robust dependent [7], i.e. in order to verify the temporal correctness of the circuit at least one of them must be considered for delay testing. But none of them can be checked alone for the $1 \to 0$ transition, since both paths are always active at the same time for this transition. To solve this problem, we will not argue on the basis of checking correctness of paths, but instead use *path systems* [15]. Thus, we can prove that the system consisting of both paths switches in time, which is sufficient for the circuit to function correctly.

III. Proofs

As illustrated by Example 3, our methodology to prove Theorem 3 is based on testing of path systems instead of individual paths. We will proceed in two steps: In the first step (Section III-A) we will associate a path system S with every input vector v in $MAXF_f \cup MINT_f$, and show that it is sufficient to 'test these path systems robustly' in order to prove that the circuit is free of delay faults. In the second step (Section III-B) it will be shown that the path system of input vector $v \in MAXF_f \cup MINT_f$, can be tested robustly by the two pattern sequence $< min_f, v >$ ($< max_f, v >$) if $v \in MINT_f$ ($v \in MAXF_f$).

A. Characterization of 'sufficient path system set'

Consider a circuit F realizing function f. For a given input vector $v \in \mathbf{B}^n$ we ask for a subcircuit S of F which can stabilize the PO on its stable (final) value $f(v)$ independent of the circuitry of F not included in S. Such a subcircuit will be called a *stabilizing system* of F for input vector v. (Note, that a stabilizing system usually only constitutes a small portion of the overall circuit. For example to stabilize the output of an OR gate to logic 1 it is sufficient to stabilize one of its inputs to 1.) Stabilizing systems are of interest since it is sufficient to bound their switching speed by the clock period τ in order to guarantee the correct operation of an implementation of F.

For a given input vector v there can exist many different stabilizing systems. Such a system will be denoted by S, and can be computed as follows:

Algorithm 1:
/* computes stabilizing system S to $v \in \mathbf{B}^n$ */
Include the PO of F and the line connecting to it in S.
While there exists a gate, fan-out node, or PI of F which is not included in S and drives a line already belonging to S:
(1) If g is a fan-out node:
Include g and its input line in S.
(2) If g is an AND, OR, NAND, NOR gate:
(a) If the stable input values of g under v are all non-controlling[3], then include g and all of its input lines in S.
(b) If a set $L = \{l_1, \ldots, l_k\}$, $k \geq 1$, of the input lines of g has controlling stable values under v, then include g and an arbitrary non empty subset[4] of L in S.
(3) If g is a NOT gate:
Include g and its input line in S.
(4) If g is a PI: Include g in S.

Definition 6:

- A *stabilizing system* of circuit F for input vector v is a triple $S = (F_S, PI_S, in_S)$, where F_S is a subcircuit of F computed by Algorithm 1, PI_S is the set of primary inputs included in F_S, and in_S is the assignment of values to the primary inputs of PI_S under

[3]The non-controlling value for an AND, NAND (OR, NOR) gate is 1 (0). The controlling value is the complement of the non-controlling one.

[4]The definition of stabilizing systems given here is more general than the one given in [30]. In [30] exactly one element of L is included in the stabilizing system, i.e. only minimal stabilizing systems are considered.

input vector v.

- S is called *maximal* for v iff F_S is obtained by always picking the complete set L in step 2(b) of Algorithm 1.

Usually, there are many possible stabilizing systems to v depending on which inputs of set L are selected in Step 2(b) if $|L| \geq 2$. The maximal stabilizing system is uniquely defined, since all inputs from L are picked here. Also note, that for any stabilizing system S the assignment of stable values to the lines of F_S is uniquely determined by input vector in_S independent of the values assigned to primary inputs not included in PI_S.

Example 4: Consider the circuit F_2 of Figure 2 under input vector $v = (11011)$. The corresponding stable value assignment is shown in Figure 3. If Algorithm 1 is applied to compute a stabilizing systems to v, then Case 2(b) with $|L| \geq 2$ only occurs for the gate marked g. Thus, there are exactly three possible stabilizing systems for this input vector, depending on whether the left, right, or both inputs of g are selected. The maximal and one non-maximal system are drawn in bold lines in Figure 3 and Figure 4, respectively. For these stabilizing systems we have $PI_S = \{x_3\}$ and in_S assigns 0 to x_3.

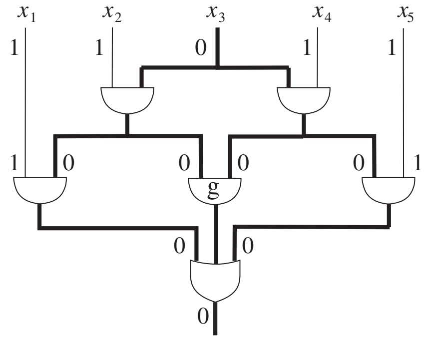

Fig. 3. Maximal stabilizing system

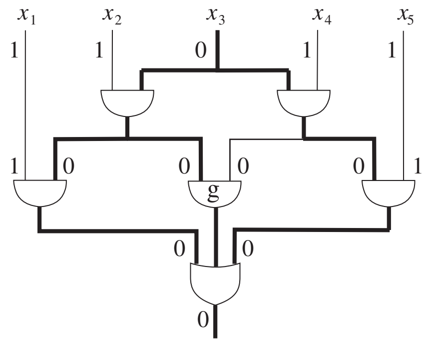

Fig. 4. Non-maximal stabilizing system

Consider an implementation $I = (F, \Delta)$ of function f. Let $v \in \mathbf{B}^n$ be an input vector to F, and $S = (F_S, PI_S, in_S)$ a stabilizing system for v. The final value $f(v)$ under input vector v can be stabilized at the output of F by only switching the elements of F_S. Thus, we can bound the delay of I for input vector v by bounding the delay of F_S in I for input in_S. The formal statement of this idea is given by the following definition and lemma.

Definition 7: Let $I = (F, \Delta)$ be an implementation of function f, and $S = (F_S, PI_S, in_S)$ a stabilizing system in F. Assume, that in_S is applied to the primary inputs of PI_S at time 0. The *guaranteed stable time of S in I ($\delta^I_{gst}(S)$)* is the earliest time such that:

> The output of I is guaranteed to be stable on its final value at time $\delta^I_{gst}(S)$ due to the circuit elements in F_S, independent of the circuitry not included in F_S and the initial values of lines in F_S at time 0.

Example 5: Consider an implementation I of example circuit F_2 with a delay assignment Δ as given in Figure 5. The numbers in brackets specify the (input pin)-to-(output pin) delay for each gate. For simplicity, we assume that this delay does not depend on the pin or the type of transition. All lines are assumed to have zero delay.

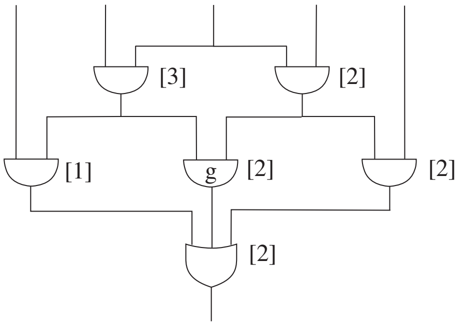

Fig. 5. Delay assignment for example circuit F_2

For this implementation of F_2 the stabilizing system S_1 of Figure 3 (S_2 of Figure 4) has a guaranteed stable time of $\delta^I_{gst}(S_1) = 6$ ($\delta^I_{gst}(S_2) = 7$). The guaranteed stable time is higher for S_2, since in this system the output of gate g has to be stabilized to zero over its left input, which takes $3 + 2 = 5$ time units. In contrast, for stabilizing system S_1 the output of gate g can be stabilized to zero over either one of its inputs, which can be achieved in $Min\{3 + 2, 2 + 2\} = 4$ time units.

Form the definitions of stabilizing system and guaranteed stable time it immediately follows:

Lemma 1: Let $I = (F, \Delta)$ be an implementation of function f. Assume that there exists a set $\mathcal{S}$ of stabilizing systems for F such that:

(1) $\forall v \in \mathbf{B}^n \, \exists S \in \mathcal{S} :\; S$ is a stabilizing system for v

(2) $\forall S \in \mathcal{S} :\; \delta^I_{gst}(S) \leq \tau$

Then, I is free of delay faults for all clock periods $\geq \tau$.

In order to apply Lemma 1 we have to solve two problems: (1) We have to find a small set $\mathcal{S}$ of stabilizing systems which 'covers all input vectors'. (2) Tests must be derived which robustly measure the delay of the systems from $\mathcal{S}$, thus bounding the guaranteed stable time for these systems. The following theorem shows how to solve the first problem for UGN realizations. The second problem will be considered in Section III-B.

Theorem 4: Consider a UGN realization F of function f. Let $\mathcal{S}$ be a set of stabilizing systems for F such that:
$\mathcal{S}$ contains a stabilizing system for every input vector from $MAXF_f \cup MINT_f$.
Then, $\mathcal{S}$ contains a stabilizing system for every input vector $v \in \mathbf{B}^n$.

In order to prove Theorem 4 we first need the following simple lemma:

Lemma 2: Let $S = (F_S, PI_S, in_S)$ be a stabilizing system. Consider an arbitrary path p in F_S going from line l' to line l''. Let us denote the stable value of a line l under some input vector v as $v(l)$, i.e. $in_S(l')$ ($in_S(l'')$) is the stable value of l' (l'') under input vector in_S. Then:

$$in_S(l'') = in_S(l') \oplus IP(p),$$

where $\oplus$ is the Exclusive-Or operation, and $IP(p)$ denotes the inversion parity of path p (see Definition 4).

Proof: Consider Algorithm 1 which defines the structure of F_S. Let g denote a gate with its output l_o element of F_S and let $l_1, \dots, l_k$ denote the inputs of g which are picked by Algorithm 1. From steps (2)(a),(b) and step (3) of Algorithm 1 it follows that for any $j \in \{1, \dots, k\}$:

$$in_S(l_o) = \begin{cases} in_S(l_j) & \text{if } g \in \{\text{AND, OR}\} \\ \overline{in_S(l_j)} & \text{if } g \in \{\text{NAND, NOR, NOT}\} \end{cases}$$

■

Example 6: Circuit F_2 given in Figure 2 only contains non-inverting gates. In this case $IP(p) = 0$ for every path p, and Lemma 2 implies that all lines of F_S have the same stable value under input vector in_S. As an example, for the stabilizing systems shown in Figures 3 and 4 all lines have stable value zero.

Proof: (of Theorem 4)
For $S = (F_S, PI_S, in_S)$ to be a stabilizing system for v, it is sufficient that v assigns in_S to the primary inputs of PI_S. Thus, we only have to show that for every $v \in \mathbf{B}^n$ there exists an $S \in \mathcal{S}$ such that for all $i \in PI_S$: $v(i) = in_S(i)$.

Without loss of generality let v be a false vertex of f, the argumentation for true vertices is analogous. From the definition of $MAXF_f$ it follows that there exists a vertex $w \in MAXF_f$ such that $w \geq_f v$. From the assumption of Theorem 4 it follows that there exists a stabilizing system $S = (F_S, PI_S, in_S)$ to w in $\mathcal{S}$. We will show that S fulfills the desired property, i.e. for all $i \in PI_S$ it holds that $v(i) = in_S(i)$:
Consider an arbitrary $i \in PI_S$. Let p be a path in F_S going from i to the PO o of F. From Lemma 2, the fact that w is a false vertex, and Observation 2 it follows that:

$$\begin{aligned} w(i) &= in_S(i) = in_S(o) \oplus IP(p) \\ &= 0 \oplus \pi_i = \pi_i \end{aligned}$$

Combining this with the fact that $w \geq_f v$ yields that $v(i) = \pi_i$. (Remember that π_i is the minimum element in component i for partial order $\geq_f$ (see Observation 1). Thus, if $w(i) = \pi_i$ and $w \geq_f v$ it follows that $v(i) = \pi_i$.)

■

Example 7: Consider the maximal false vertex $w = (11011)$ for our example circuit F_2 realizing function $f = x_1x_2x_3 \vee x_2x_3x_4 \vee x_3x_4x_5$. Since $f(w) = 0$ and $\pi_i = 0$ for all $i \in \{1, \dots, 5\}$, it follows from Lemma 2 that any stabilizing system S to w only includes PIs which are set to 0, i.e. $PI_S = \{x_3\}$. For an arbitrary input vector v with $w >_f v$ variable x_3 is also set to 0. Thus, any stabilizing system for w is also a stabilizing system for v. Figure 6 illustrates this fact for input vector $v = (01000)$ and the stabilizing system for $w = (11011)$ of Figure 3.

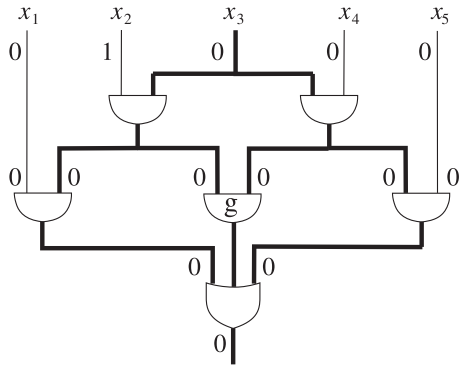

Fig. 6. Stabilizing system for (01000)

B. Measuring guaranteed stable times

From the preceding section it follows that it is sufficient to robustly measure the delay of one stabilizing system S_v for every input vector $v \in MINT_f \cup MAXF_f$ in order to prove that a UGN implementation of f is free of delay faults. The purpose of this section is to choose S_v, and show that the test sequence $< min_f, v >$ ($< max_f, v >$) robustly measures the guaranteed stable time of S_v for $v \in MINT_f$ ($v \in MAXF_f$).

The following lemma gives a method to robustly measure the guaranteed stable time of a stabilizing system. In order to formulate it, it is useful to recall the definitions of earliest arrival time [42] and off-path inputs [1]:

Definition 8: Assume that a circuit implementation $I = (F, \Delta)$ has stabilized under some input vector w. At time 0 the circuit inputs are switched from w to v. Let $v(l)$ denote the stable value of some line l under input vector v.

> The *earliest arrival time* of $v(l)$ on line l under transition $w \rightarrow v$ is the first instant of time at which l is set to $v(l)$. It is denoted by $EAT^I_{w\rightarrow v}(l)$.

Definition 9: Let $S = (F_S, PI_S, in_S)$ be a stabilizing system, g a gate such that some inputs of g are included in F_S, and l an arbitrary input line to g.

$$l \text{ is called } \textit{on-path (off-path)} \text{ input of } S \iff l \in F_S \quad (l \notin F_S).$$

Lemma 3: Let F be an arbitrary circuit, and $S = (F_S, PI_S, in_S)$ a stabilizing system in F. Consider a test sequence $< w, v >$ for S with the following properties:

(T1) During the transition from w to v all off-path inputs of S are stable on non-controlling values.

(T2) $\forall i \in PI_S: \ v(i) = in_S(i) = \overline{w(i)}$.

If $< w, v >$ fulfills properties (T1) and (T2), then for the primary output o of any implementation $I = (F, \Delta)$ of F it holds that:

$$EAT^I_{w\rightarrow v}(o) = \delta^I_{gst}(S)$$

Proof: The proof directly follows from the following remarks:
Remark 1: Because of property (T1) we know that the output value under v can only be reached due to the circuitry included in F_S.
Remark 2: From the definition of stabilizing system and the fact that all side inputs are non-controlling it directly follows that, if we flip all input bits to F_S, also all values of internal lines in F_S will be flipped. Thus, for all lines l in F_S:

$$v(l) = \overline{w(l)}$$

As a consequence, the test $< w, v >$ creates the worst case situation for switching F_S under input vector in_S.
Remark 3: From the definition of stabilizing system we conclude that all the input lines of a given gate which belong to F_S have the same transition under test sequence $< w, v >$. (Remember that for a given gate g Algorithm 1 only includes input lines with the same value into a stabilzing system.) Thus, if the primary inputs switch hazard-free for test sequence $< w, v >$, also all internal lines in F_S will switch without any hazard.

From the above remarks it directly follows that $EAT^I_{w\rightarrow v}(o) = \delta^I_{gst}(S)$. (The formal proof, which is done by structural induction, is omitted here for brevity.) ∎

Example 8: Consider the maximal stabilizing system S_1 for input vector (11011) given in Figure 3. For this system, $PI_{S_1} = \{x_3\}$ and in_{S_1} sets x_3 to 0. The test sequence $< (11111), (11011) >$ fulfills conditions (T1) and (T2) and thus, can be used to measure the guaranteed stable time of S_1. The first input vector (11111) initializes all lines of F_{S_1} to stable 1. Since the stable value under input vector in_{S_1} is 0 for all these lines, this initialization creates the worst case starting point for switching S_1. The second vector (11011) only flips input variable x_3 and thus, guarantees that all off-path inputs of S_1 are stable on non-controlling values. As a consequence, if logic 0 is observed at the primary output of the circuit, this can only be due to the switching of F_{S_1}.

As an example, consider the implementation of F_2 given in Figure 5. The earliest arrival time of value 0 at the PO of the circuit under test sequence $< (11111), (11011) >$ is 6 time units. It equals the guaranteed stable time of S_1 for this implementation.

Lemma 3 naturally generalizes the notion of a robust test for a path delay fault. To see this, assume that F_S only consists of a single path p. Then, conditions (T1) and (T2) are exactly the conditions for a robust test of path p under transition $\overline{in_S} \rightarrow in_S$ at its input.[5]

Definition 10: Consider a circuit F and let $S = (F_S, PI_S, in_S)$ be a stabilizing system in F. A test sequence $< w, v >$ is called a *robust test* for S iff it fulfills properties (T1) and (T2) of Lemma 3.

We still have to answer the question whether a robust test can be found for any stabilizing system S.

Example 9: Consider the non-maximal stabilizing system S_2 for input vector (11011) given in Figure 4. S_2 is not robustly testable, since condition (T1) cannot be fulfilled. The second vector of the test sequence must set $x_3 = 0$. This forces the right input of gate g which is an off-path input for S_2 to the controlling value 0.

In contrast, the corresponding maximal stabilizing system S_1 considered in Example 8 is robustly testable. This is due to the fact that in computing a maximal stabilizing system, all input lines with controlling stable values are chosen in Step (2)(b) of Algorithm 1 (see Definition 6). Thus, the problematic right input line of gate g is no off-path input for S_1.

The following theorem shows that for UGN realizations maximal stabilizing systems can always be tested robustly as described in Example 8.

Theorem 5: Consider a unate gate network F realizing function f. Let S be the maximal stabilizing system for a vertex v, and $w := min_f$ ($w := max_f$) if $v \in TRUE_f$

[5] If F_S consists only of a single path p, then by Algorithm 1 all stable values of on-path inputs to AND, OR, NAND, NOR gates must be controlling for input in_S at the PI of p. Thus, condition (T1) is not too restrictive.

($v \in FALSE_f$). Then, the test sequence $< w, v >$ is a robust test for S.

Proof: Since the proof of this theorem is quite technical, it is defered to the Appendix. ∎

From Lemma 1 and Theorems 4, 5 it now easily follows that:

$TT_f = (\{min_f\} \times MINT_f) \cup (\{max_f\} \times MAXF_f)$ is a universal delay fault test set for arbitrary UGN implementations of function f (Theorem 3).

Proof: (of Theorem 3)
Let $I = (F, \Delta)$ be an arbitrary UGN implementation of function f which has passed the tests from TT_f. Consider the set $\mathcal{S} := \{S | S$ is a maximal stabilizing system to a vertex $v \in MINT_f \cup MAXF_f$ in $F\}$. Since I has passed the tests from TT_f for clock period τ, we conclude from Theorem 5 that for all $S \in \mathcal{S}$: $\delta^I_{gst}(S) \leq \tau$. From Theorem 4 it follows that for every input vector $v \in \mathbf{B}^n$ there exists a stabilizing system $S \in \mathcal{S}$. Application of Lemma 1 then immediately yields that I is free of delay faults. ∎

IV. Generalization to arbitrary functions

Let f' be an arbitrary function which is binate in variables $x_1, \ldots, x_b$ and unate in variables $x_{b+1}, \ldots, x_n$. It is well known [39], [25] that there always exists a unate function $f : \mathbf{B}^{n+b} \rightarrow \mathbf{B}$ in variables $y_1, y'_1, \ldots, y_b, y'_b, y_{b+1}, \ldots, y_n$ which is positive (negative) unate in variables $y_1, \ldots, y_b$ ($y'_1, \ldots, y'_b$) such that:

$$f(x_1, x_1, \ldots, x_b, x_b, x_{b+1}, \ldots, x_n) = f'(x_1, \ldots, x_n)$$

Thus, f' can be computed with the help of a UGN F realizing function f. As indicated in Figure 7, the network G which drives the input register to F could again be a unate gate network which fans out the binate variables of f'. (Clearly, such an output fan-out does not destroy the UGN property of G.)

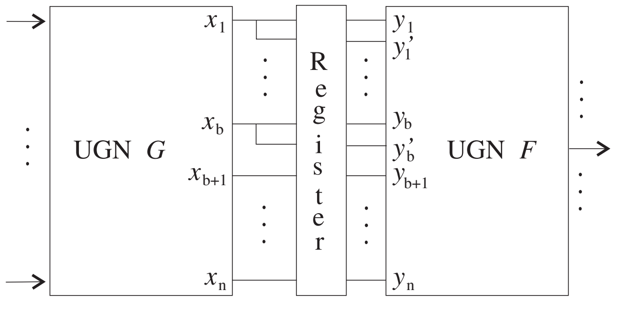

Fig. 7. System built of UGNs

There are several ways to obtain the unate gate network F:

(1) Any given non-UGN realization F' of f' can be transformed into a UGN F such that $F(x_1, x_1, \ldots, x_b, x_b, x_{b+1}, \ldots, x_n) = F'(x_1, \ldots, x_n)$ [25], [22]. This 'UGN-transformation' guarantees that:

- The delay of F is not larger than the delay of F'.
- The size of F is at most double the size of F'.

Figures 8 and 9 show the result of this transformation for a simple example circuit. Note, that there is a one to one correspondence between the paths of both circuits. As a result, the UGN-transformation does not increase the delay of a circuit.

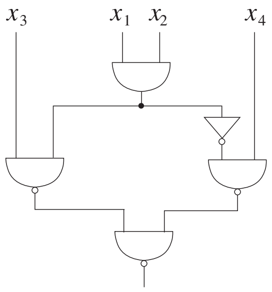

Fig. 8. Non-unate circuit F'

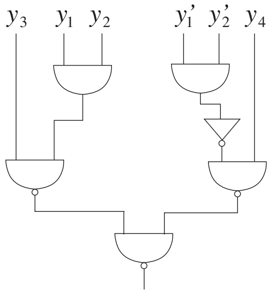

Fig. 9. Transformed unate circuit F

(2) F can be synthesized with a constrained logic synthesis tool. Methods to constrain the MIS logic synthesis system [43] in order to produce UGN realizations have been studied in the context of on-line error detection with Berger codes [35], [36].

The problem with UGN realizations of arbitrary functions as shown in Figure 7 is, that not all input combinations can be applied to F, i.e. for an input pair (y_i, y'_i), $i \in [1 : b]$, we have $(y_i, y'_i) \notin \{01, 10\}$. As a consequence, the universal test set for function f as given by Theorem 3 is not applicable.

This problem can be solved by transforming the input register of F in Figure 7 in such a way that all test sequences of the form $< min_f, v >$ and $< max_f, v >$, $v \in \mathbf{B}^{n+b}$, can be applied to F. In order to do so, we need a scan-chain[6] to load an arbitrary vector $v \in \mathbf{B}^{n+b}$, and a set and reset capability for the output latches of the chain to generate the initialization patterns min_f and max_f. (The corresponding hardware modifications for the LSSD scan-chain have been given in [22]. Note, that adding set and reset is cheaper than modifying the scan-chain such that arbitrary two pattern delay tests can be applied [22].)

V. Experimental Results

To evaluate the applicability of the theory presented in the previous sections, we conducted experiments on two sets of circuits. The first group of examples considered includes all ISCAS89 benchmarks [37] up to circuit cs1494. The second group consists of synthesized multi-level implementations[7] for the two-level MCNC benchmarks [38]. From these example circuits only those were considered which are not completely path delay testable.

To explore the hardware overhead induced by realizing a circuit as a unate gate network, we transformed each of the circuits into a UGN as given in Section IV (Method (1)). Table I (Table II) shows the results obtained for the first group (a representative sample of the second group) of examples. The name of the circuit is given in the first column. The second (third) column lists the circuit size[8] for the original (transformed) circuit. The last column gives the percentage increase in circuit size of the UGN compared to the original circuit. Delays are not given, since they are the same for the original and the transformed circuit.

As can be seen from these tables, the hardware overhead varies extremely with the circuit under consideration. (From 1.4% for circuit cs832 in Table I to 88.5% for circuit t481 in Table II.) For several examples the overhead for designing them as UGNs is in reasonable limits. Thus, for more than one third of all examples considered the increase in circuit size is less than 15%. In addition, it should be noted that designing an optimal non-unate circuit and then transforming this circuit into a UGN, will usually not give the best possible UGN realization. Thus, the gate overheads given in Tables I and II are upper bounds on the actual overheads implied by restricting to unate gate network realizations. Also, for circuits which are naturally designed as UGNs (like dynamic CMOS logic or self-checking circuits based on unordered codes) the test methodology developed in this paper can be applied without inducing any gate overhead.

Tables III and IV show the results of test generation for the two sets of example circuits. Since there is a one to one

[6]By adapting the test set to the given input restrictions it is also possible to omit the scan chain at the cost of a possible increase in test set size [44].

[7]Each circuit was synthesized by applying script.rugged in the SIS system [45].

[8]The cost of an inverter, buffer (Nand, Nor) (And, Or) has been calculated as 0.5 ($\#inputs \cdot 0.5$) ($\#inputs \cdot 0.5 + 0.5$).

TABLE I
Circuit sizes for ISCAS89 benchmarks

circuit	original	UGN	% gate increase
cs27	10.5	18.5	76.2 %
cs298	145.5	163	12.0 %
cs344	161	200.5	24.5 %
cs349	163.5	204	24.8 %
cs382	170.5	204.5	19.9 %
cs386	232.5	238	2.4 %
cs400	178	219	23.0 %
cs444	189.5	230.5	21.6 %
cs510	243.5	258.5	6.2 %
cs526	264.5	289.5	9.5 %
cs641	321	444.5	38.5 %
cs713	351	545	55.3 %
cs820	446.5	453	1.5 %
cs832	455.5	462	1.4 %
cs953	414	434	4.8 %
cs1196	614	820.5	33.6 %
cs1238	643.5	858	33.3 %
cs1423	749	1026.5	37.0 %
cs1488	968.5	1014	4.7 %
cs1494	975.5	1021	4.7 %

TABLE II
Circuit sizes for MCNC benchmarks

circuit	original	UGN	% gate increase
9sym	226.5	251.5	11.0 %
apex2	225.5	255.5	13.3 %
apex3	1357	1619	19.3 %
apex5	663.5	825	24.3 %
clip	129	192	48.8 %
cps	970.5	1087	12.0 %
duke2	345	415	20.3 %
ex4	663.5	692.5	4.4 %
ex5	413.5	472	14.1 %
misex1	46	82.5	79.3 %
misex2	83.5	109.5	31.1 %
misex3	559	840	50.3 %
misex3c	417.5	739.5	77.1 %
pdc	335	419	25.1 %
rd73	62	107.5	73.4 %
sao2	128.5	218.5	70.0 %
spla	535	607	13.5 %
t481	30.5	57.5	88.5 %
vg2	84	87.5	4.2 %
Z9sym	215.5	222	3.0 %

TABLE III
Test results for ISCAS89 benchmarks

circuit	# LP	% rt LP	# PDTS	# UTS
cs27	56	89.3 %	16	15
cs298	462	74.2 %	201	126
cs344	710	86.1 %	392	2798
cs349	730	83.7 %	386	2798
cs382	800	83.4 %	358	590
cs386	414	99.8 %	183	121
cs400	896	74.0 %	335	587
cs444	1070	54.8 %	342	587
cs510	738	98.8 %	359	628
cs526	820	84.6 %	383	310
cs641	3488	56.7 %	815	5089
cs713	43624	2.7 %	482	5463
cs820	984	99.6 %	425	895
cs832	1012	97.2 %	417	885
cs953	2312	99.6 %	938	2065
cs1196	6196	57.8 %	1664	18324
cs1238	7118	50.4 %	1534	15939
cs1423	89452	31.6 %	14685	†
cs1488	1924	97.5 %	818	1572
cs1494	1952	96.4 %	844	1531

TABLE IV
Test results for MCNC benchmarks

circuit	# LP	% rt LP	# PDTS	# UTS
9sym	1222	62.8 %	693	2394
apex2	2860	73.6 %	1366	4800
apex3	35270	24.3 %	4175	8274
apex5	23848	35.0 %	4739	4995
clip	2694	22.0 %	327	1674
cps	12608	89.0 %	3468	8207
duke2	5972	47.5 %	1062	4723
ex4	2060	96.8 %	1594	734
ex5	8318	25.0 %	750	1647
misex1	1286	17.1 %	86	93
misex2	494	74.3 %	196	119
misex3	40578	22.7 %	4071	15972
misex3c	1856452	na[10]	na	36271
pdc	3544	72.8 %	1409	4072
rd73	4116	15.0 %	390	2574
sao2	5076	10.4 %	293	736
spla	6274	69.8 %	2370	3328
t481	128	100 %	110	†
vg2	914	86.2 %	470	1213
Z9sym	852	91.2 %	694	1066

correspondence between the paths in the original and the transformed circuit, the number of logical paths[9] (# LP) is given only once in column two. Columns three and four list the percentage of robustly testable logical paths (% rt LP) and the number of test sequences in the corresponding path delay test set (# PDTS) for the original circuits. The corresponding test sets were computed by the test generator presented in [2] with the backtrack limit set to 1000. The last column gives the number of test sequences for the universal test set (# UTS) as given by Theorem 3. Entry '†' in this column indicates that the size of the UTS was too large to be acceptable. The test sets were obtained by computing the UTS for each output cone separately, and then combining these test sets based on the technique described in [29] in order to minimize the size of the overall test set.

Again, the results very much depend on the circuit under consideration. For most of the designs the universal test set for the transformed circuit is larger than the path delay test set for the original circuit. This is not surprising, since the UTS constitutes a complete delay test whereas the path delay test sets only check the circuits partially. Compared to its high test quality the size of the universal test set is reasonable for most of the designs. 28 out of the 40 circuits have a UTS of size less than the total number of logical paths. For circuits misex1 (misex3c) the size of the UTS is even only 7.2% (2%) of the overall number of logical paths.

There are also two designs (marked '†' in column four) for which the size of the UTS exceeds the number of logical paths extremely. As an example, for circuit t481 of Table IV the number of logical paths is only 128 whereas the size of the UTS is 6969. The existence of such circuits is not surprising because of the following well known fact [39]: the minimal true vertices (maximal false vertices) of a unate function f correspond to the prime implicants of f ($\overline{f}$). Thus, for functions where the number of prime implicants of f or $\overline{f}$ is extremely large, testing with universal test sets becomes too expensive.

Even for a circuit like misex3c, where the size of the UTS $(36,271)$ is much less than the overall number of logical paths $(1,856,452)$, the universal test set might still be considered too large for practical purposes. In path delay testing one common approach to handle such circuits with very large path numbers is to only check those paths with expected delay greater than a given threshold t [46]. A similar methodology can also be applied for testing stabilizing systems, i.e. only those test sequences of the UTS are selected for testing purposes which check the

[9] A logical path is a physical path leading from a primary input to a primary output together with a transition (either $0 \rightarrow 1$ or $1 \rightarrow 0$) at its input.

[10] This circuit could not be handled by the test generator of [2] because of its large number of paths.

stabilizing systems with expected delay greater than t in the given design. Thus, in case that the overall size of the UTS is too large to be accecptable, the universal test set can be tailored to a given design by restricting to those test sequences which check the most critical parts of this design.

VI. Conclusion

In this paper we have studied the theory of universal test sets for logic functions. The universal test set for a given function f is valid for any UGN implementation of f. It has been proven that such a universal test set is much more powerful than implied by previous results, i.e. besides checking for stuck-at and stuck-open faults it also guarantees the correctness of the circuit with respect to arbitrary delay faults.

Our results are of special interest in application areas where circuits are naturally built as UGNs, i.e. for the design of dynamic CMOS logic or self-checking circuits based on unordered codes. Another important application is given by the observation that any circuit can be transformed into a UGN with the same delay and at most double the size of the original network. This implies a design for delay-testability method which does not compromise the speed of a given circuit realization. With respect to self test methods for delay faults it follows from the results of this paper that for UGNs we can restrict to only two initialization vectors (min_f and max_f). Thus, test time can be reduced, and the structure of the random pattern generator can be simplified. Experimental results have shown that for many functions there exist efficient UGN implementations, and that the corresponding universal test sets are of reasonable size.

Appendix: Proof of Theorem 5

Consider a unate gate network F realizing function f. Let S be the maximal stabilizing system for an input vector v, and $w := max_f$ ($w := min_f$) if $v \in FALSE_f$ ($v \in TRUE_f$).

For proving Theorem 5 we have to show that the test sequence $< w, v >$ is a robust test for S.

To do so, we first prove two simple lemmas which are needed to argue properties (T2) and (T1) of the definition of robust test.

Lemma 4: Consider a test sequence $< w, v >$ such that $w = max_f$ and $v \in FALSE_f$ ($w = min_f$ and $v \in TRUE_f$). Let $S = (F_S, PI_S, in_S)$ be a stabilizing system to v, then:

$$\forall i \in PI_S : \ v(i) = \overline{w(i)}$$

Proof: We will only prove the lemma for $v \in FALSE_f$ and $w = \max_f$. The case $v \in TRUE_f$ and $w = \min_f$ is handled analogously.

Since v is a false vertex, and $i \in PI_S$ it follows from Lemma 2 that: $0 = v(i) \oplus \pi_i$, i.e. $v(i) = \pi_i$. The claim then follows from the definition of $w = max_f = (\overline{\pi_1}, \ldots, \overline{\pi_n})$. ■

Lemma 5: Consider a test sequence $< w, v >$ such that $w = max_f$ and $v \in FALSE_f$ ($w = min_f$ and $v \in TRUE_f$). Let S be the maximal stabilizing system to v and l an arbitrary off-path input of S. For the maximal stabilizing system $S_l = (F_{S_l}, PI_{S_l}, in_{S_l})$ of line l under input vector v we have:

$$\forall i \in PI_{S_l} : \ v(i) = w(i)$$

Proof: Again without loss of generality we will only prove the case $v \in FALSE_f$, $w = max_f$. Since $max_f = (\overline{\pi_1}, \ldots, \overline{\pi_n})$, we have to show that $v(i) = \overline{\pi_i}$ $\forall i \in PI_{S_l}$.

Let g be the gate with input l. The input of g which belongs to F_S is denoted by l' and the output of g by l_o. We know that there exists a path p_1 in F_{S_l} from some input i of F to l, and a path p_2 in F_S from output l_o of g to the output o of F. The corresponding situation is depicted in Figure 10.

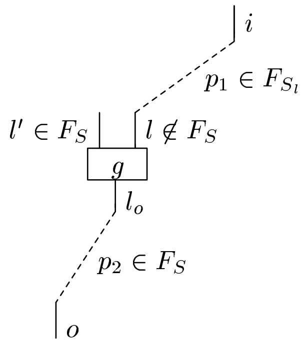

Fig. 10. Configuration at gate g

Let us denote the parity of gate g by π_g, i.e. $\pi_g = 0$ ($\pi_g = 1$) iff g is an AND, OR (NAND, NOR, NOT) gate. From the fact that l' and l_o belong to F_S it follows by Lemma 2 that $v(l_o) = v(l') \oplus \pi_g$. In addition, since l does not belong to F_S, we conclude that $v(l) = \overline{v(l')}$. (If $v(l) = v(l')$, then Algorithm 1 would have picked both inputs for F_S, since S is maximal.) Thus, $v(l) = 1 \oplus v(l') = 1 \oplus \pi_g \oplus v(l_o)$. Besides the above equality, it follows from Lemma 2 that $v(i) = IP(p_1) \oplus v(l)$ ($v(l_o) = IP(p_2) \oplus v(o) = IP(p_2) \oplus 0$), since p_1 (p_2) is a path in F_{S_l} (F_S). Putting all these relations together we obtain:

$$\begin{aligned} v(i) &= IP(p_1) \oplus v(l) \\ &= IP(p_1) \oplus 1 \oplus \pi_g \oplus v(l_o) \\ &= IP(p_1) \oplus 1 \oplus \pi_g \oplus IP(p_2) \oplus 0 \\ &= IP(p_1) \oplus \pi_g \oplus IP(p_2) \oplus 1 \end{aligned}$$

Combining this with the fact that $IP(p_1) \oplus \pi_g \oplus IP(p_2) = \pi_i$ (see Observation 2) finishes the proof. ■

Theorem 5 now follows directly from Lemmas 4 and 5.

Proof: (of Theorem 5)
Property (T2) follows directly from Lemma 4. From the definition of a maximal stabilizing system we conclude that all off-path inputs of S have non-controlling values under input vector v. From Lemma 5 it follows that these non-controlling values are stable during the transition from w to v, i.e. property (T1) is fulfilled too. ■

Acknowledgment

The authors would like to thank Prof. G. Hotz for his support and many valuable discussions.

References

[1] C.J. Lin and S.M. Reddy, "On delay fault testing in logic circuits," *IEEE Transactions on CAD*, pp. 694–703, 1987.

[2] S.M. Reddy, C.J. Lin, and S. Patil, "An automatic test pattern generator for the detection of path delay faults," in *Proceedings of the International Conference on Computer-Aided Design*, 1987, pp. 284–287.

[3] V.S. Iyengar, B.K. Rosen, and I. Spillinger, "Delay test generation 2 – algebra and algorithms," in *Proceedings of the International Test Conference*, 1988, pp. 867–876.

[4] K. Fuchs, F. Fink, and M.H. Schulz, "DYNAMITE: an efficient automatic test pattern generation system for path delay faults," *IEEE Transactions on CAD*, 1991.

[5] E.S. Park and M.R. Mercer, "An efficient delay test generation system for combinational logic circuits," *IEEE Transactions on CAD*, pp. 926–938, 1992.

[6] W. Mao and M.D. Ciletti, "Robustness enhancement and detection threshold reduction in ATPG for gate delay faults," in *Proceedings of the International Test Conference*, 1992, pp. 588–597.

[7] W.K. Lam, A. Saldanha, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "Delay fault coverage and performance tradeoffs," in *Proceedings of the 30th Design Automation Conference*, 1993, pp. 446–451.

[8] K.-T. Cheng and H.-C. Chen, "Delay testing for non-robust untestable circuits," in *Proceedings of the International Test Conference*, 1993, pp. 954–961.

[9] U. Mahlstedt, "DELTEST: Deterministic test generation for gate delay faults," in *Proceedings of the International Test Conference*, 1993, pp. 972–980.

[10] I. Pomeranz, S.M. Reddy, and P. Uppaluri, "NEST: A non-enumerative test generation method for path delay faults in combinational circuits," in *Proceedings of the 30th Design Automation Conference*, 1993, pp. 439–445.

[11] S. Bose, P. Agrawal, and V.D. Agrawal, "Logic systems for path delay test generation," in *Proceedings of the EURO-DAC*, 1993.

[12] K. Fuchs, M. Pabst, and T. Rössel, "RESIST: A recursive test generation algorithm for path delay faults considering various test classes," *IEEE Transactions on CAD*, pp. 1550–1562, 1994.

[13] Z. Barzilai and B.K. Rosen, "Comparision of AC self-testing procedures," in *Proceedings of the International Test Conference*, 1983, pp. 89–91.

[14] G.L. Smith, "Model for delay faults based upon paths," in *Proceedings of the International Test Conference*, 1985, pp. 342–349.

[15] W. Ke and P.R. Menon, "Synthesis of delay-verifiable two-level circuits," in *Proceedings of the European Design and Test Conference*, 1994, pp. 297–301.

[16] S. Kundu and S.M. Reddy, "On the design of robust testable CMOS combinational logic circuits," in *Proceedings of the International Symposium on Fault-Tolerant Computing*, 1988, pp. 220–225.

[17] K. Roy, K. De, J.A. Abraham, and S. Lusky, "Synthesis of delay fault testable combinational logic," in *Proceedings of the International Conference on Computer-Aided Design*, 1989, pp. 418–421.

[18] A.K. Pramanick and S.M. Reddy, "On the design of path delay fault testable combinational circuits," in *Proceedings of the International Symposium on Fault-Tolerant Computing*, 1990, pp. 374–381.

[19] M.J. Bryan, S. Devadas, and K. Keutzer, "Testability preserving circuit transformations," in *Proceedings of the International Conference on Computer-Aided Design*, 1990, pp. 456–459.

[20] S. Devadas and K. Keutzer, "Synthesis and optimization procedures for robustly delay-fault testable logic circuits," in *Proceedings of the 27th Design Automation Conference*, 1990, pp. 221–227.

[21] I. Pomeranz and S.M. Reddy, "Achieving complete delay fault testability by extra inputs," in *Proceedings of the International Test Conference*, 1991, pp. 273–282.

[22] A.K. Pramanick and S. Kundu, "Design of scan-based path delay testable sequential circuits," in *Proceedings of the International Test Conference*, 1993, pp. 962–971.

[23] H. Hengster, R. Drechsler, and B. Becker, "Testability properties of local circuit transformations with respect to the robust path-delay-fault model," in *Proceedings of the VLSI Design Conference*, 1994, pp. 123–126.

[24] A. El-Maleh and J. Rajski, "Delay-fault testablility preservation of the concurrent decomposition and factorization transformations," in *Proceedings of the 12th IEEE VLSI Test Symposium*, 1994.

[25] S.M. Reddy, "Complete test sets for logic functions," *IEEE Transactions on Computers*, pp. 1016–1020, 1973.

[26] R. Betancourt, "Derivation of minimum test sets for unate logic circuits," *IEEE Transactions on Computers*, pp. 1264–1269, 1971.

[27] S.B. Akers, "Universal test sets for logic networks," *IEEE Transactions on Computers*, pp. 835–839, 1973.

[28] G. Gupta and N.K. Jha, "A universal test set for CMOS circuits," *IEEE Transactions on CAD*, pp. 590–597, 1988.

[29] B. Chen and C.L. Lee, "A complement-based fast algorithm to generate universal test sets for multi-output functions," *IEEE Transactions on CAD*, pp. 370–377, 1994.

[30] U. Sparmann, D. Luxenburger, K.T. Cheng, and S.M. Reddy, "Fast identification of robust dependent path delay faults," in *Proceedings of the 32nd Design Automation Conference*, 1995, pp. 119–125.

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

[32] N. Goncalves and H. DeMan, "NORA: A racefree dynamic CMOS technique for pipelined logic structures," *IEEE Journal of Solid State Circuits*, 1983.

[33] L.G. Heller, W.R. Griffin, J.W. Davis, and N.G. Thoma, "Cascode voltage switch logic: A differential CMOS logic family," in *Proceedings of the IEEE International Solid State Circuits Conference*, 1984.

[34] V. Peng, D.R. Donchin, and Y.-T. Yen, "Design methodology and CAD tools for the NVAX microprocessor," in *Proceedings of the IEEE Int. Conf. on Computer Design*, 1992.

[35] K. De, C. Wu, and P. Banerjee, "Reliability driven logic synthesis of multilevel circuits," in *Proceedings of the International Symposium on Circuits and Systems*, 1992, pp. 1105–1108.

[36] N.K. Jha and S.-J. Wang, "Design and synthesis of self-checking VLSI circuits," *IEEE Transactions on CAD*, pp. 878–887, 1993.

[37] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles for sequential benchmark circuits," in *Proceedings of the International Symposium on Circuits and Systems*, 1989, pp. 1929–1934.

[38] S. Yang, "Logic synthesis and optimization benchmarks user guide (version 3.0)," Tech. Rep., Microelectronics Center of North Carolina, P.O. Box 12889, Research Triangle Park, NC 27709, January 1991.

[39] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, 1978.

[40] M.K. Reddy, S.M. Reddy, and V.D. Agrawal, "Robust tests for stuck-open faults in CMOS combinational logic citcuits," in *Proceedings of the 14th International Symposium on Fault-Tolerant Computing*, 1984, pp. 44–49.

[41] W.K. Lam, A. Saldanha, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "Delay fault coverage, test set size, and performance tradeoffs," Tech. Rep., No. UCB/ERL M92/119, University of California, Berkeley CA, 1992.

[42] V.S. Iyengar, B.K. Rosen, and I. Spillinger, "Delay test generation 1 – concepts and coverage metrics," in *Proceedings of the International Test Conference*, 1988, pp. 857–866.

[43] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple-level logic optimization system," *IEEE Transactions on CAD*, pp. 1062–1081, 1987.

[44] U. Sparmann and S.M. Reddy, "Universal delay test sets for logic networks," Tech. Rep. 5-30-94, Department of ECE, University of Iowa, Iowa City, IA 52242, 1994.

[45] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *Proceedings of the International Conference on Computer Design*, 1992, pp. 328–333.

[46] Y.K. Malaiya and R. Narayanaswamy, "Testing for timing faults in synchronous sequential integrated circuits," in *Proceedings of the International Test Conference*, 1983, pp. 560–571.