On Primitive Fault Test Generation in Non-Scan Sequential Circuits*

Ramesh C. Tekumalla
Intel Corporation
2501 NW 229th Ave.
Hillsboro, OR 97124

Prem R. Menon
Dept. of ECE
University of Massachusetts
Amherst, MA 01003

Abstract

A method is presented for identifying primitive path-delay faults in non-scan sequential circuits and generating robust tests for all robustly testable primitive faults. It uses the concept of sensitizing cubes introduced in an earlier paper and a new, more efficient algorithm for generating them. Sensitizing cubes of the next-state and output logic are used to obtain static sensitizing vectors that can be applied to the non-scan sequential circuit as part of a vector-pair. These vector-pairs are also used in deriving robust tests. Initializing sequences from a reset state and sequences that propagate fault effects from flip-flops to primary outputs are also generated. The proposed method has been implemented and used to derive tests for primitive faults in ISCAS'89 and MCNC'91 benchmark circuits.

1 Introduction

Delay testing is becoming increasingly important because of the need to operate digital circuits at the highest possible speeds. The circuit under test is assumed to have been tested for logical faults and therefore functionally correct. Faulty behavior is modeled by delay faults, the most widely used model being the path delay fault [1]. The presence of a path delay fault increases the propagation delay along the faulty path beyond the limit for correct operation. A test for a path delay fault propagates a signal transition along the path and checks whether the final value reaches the destination at the appropriate time.

Delay testing of combinational circuits has received considerable attention in the literature, e.g., [1]-[8]. Since delay faults are often the result of variations in the fabrication process, we must be able to detect individual faults independent of actual delay values in the rest of the circuit [1]. Tests that satisfy this condition are called robust tests [2]. Several techniques have been developed for generating robust tests for delay faults in combinational circuits [2, 3, 4], but the fault coverage obtained is often quite low. A weaker class of tests, called validatable non-robust (VNR) tests, has been proposed to test faults that are not robustly testable [3, 5]. A VNR test detects the target fault if a set of other faults is also tested for and shown to be absent. Even with the use of VNR tests, all faults in a circuit may not be testable.

Although a large fraction of path delay faults in a combinational circuit may not be testable, some of them may not have any effect on circuit delay. These faults are redundant, and need not be tested [6]. Another class of faults, called robust dependent (RD) faults [8] need not be tested if all remaining (non-RD) faults are tested by robust tests. If the circuit contains any untestable non-RD fault, it may malfunction even if it produces correct responses for all tests. Thus, an important problem in delay testing is to identify all faults that must be tested to guarantee that the tested circuit will function correctly at the tested speed and any lower speed.

It has been shown that, for any combinational circuit, there exists a set of faults called primitive faults, that must be tested to guarantee correct timing [9, 10]. The set of primitive faults may contain single and multiple path delay faults. A method of identifying primitive faults in two-level circuits was proposed in [9]. An extension for arbitrary multi-level circuits using its ENF [11] was also suggested, but may be applicable only to relatively small circuits. Sivaraman and Strojwas [12] have developed a method based on stabilization times applicable to multi-level circuits. It uses a different fault model based on physical parameters of the circuit being tested. Krstic, et al. [13] have proposed a method that identifies all primitive faults of cardinality less than or equal to two. It can be extended to find primitive faults of cardinality greater than two at the expense of computational complexity. A method of primitive fault identification and test generation using the concept of sensitizing cubes [14] has been shown to be effective for circuits of moderate size.

While the results on delay testing of combinational circuits are directly applicable to sequential circuits with scan, the use of enhanced-scan flip-flops to allow application of arbitrary vector pairs results in increased chip area and performance degradation. The use of special flip-flops to improve delay-fault testability with-

*This research was in part supported by NSF grant MIP-9320886

out increasing path delays has been found to complicate test application [15]. Methods of partial enhanced-scan have been proposed [16, 17], but they do not provide adequate coverage of delay faults. Methods of synthesizing testable non-scan sequential circuits have been proposed in [18, 19], but the area overhead of the methods is high. Even if scan design is used, the only faults that can affect circuit behavior are those that are sensitized during normal (non-scan) operation. While identification of redundant and untestable faults [20, 21] can be used to reduce test generation effort, the remaining set would still contain faults that need not be tested. Thus, primitive faults must be identified assuming non-scan operation to avoid unnecessary tests and pessimistic test results.

The goal of this paper is to develop methods of identifying primitive faults in non-scan sequential circuits, and deriving robust tests for robustly testable primitive faults. Combinationally primitive faults are first identified using sensitizing cubes as in [14]. However, sensitizing cubes are obtained using a new method, which is expected to be more efficient. A combinationally primitive fault is primitive in the sequential circuit if there is a vector-pair that sensitizes it can be justified, and fault effects propagated to a primary output. If no such vector-pair exists, multiple faults containing the combinationally primitive fault are analyzed in the order of increasing size. The test generation procedure produces complete test sequences, including initialization and fault propagation to primary outputs.

The rest of the paper is organized as follows. After presenting some basic definitions in Section 2, we discuss primitive fault identification in combinational circuits in Section 3. Sections 4 and 5 discuss primitive fault identification and test generation in sequential circuits. Experimental results and conclusion are in Sections 6 and 7, respectively.

2 Preliminaries

Most of the definitions that apply to single paths and single path delay faults [2, 6] can be generalized to multiple paths and multiple-path delay faults.

A set of single paths $\{\pi_1\pi_2, ...\pi_n\}$ to the same destination is called a *multipath* Π. A gate input on any single path $\pi_i \in \Pi$ is an *on-path input* of Π. An input to any gate on Π, that is not an on-path input, is a *side input* of Π.

A *multipath delay fault (MPDF)* on Π consists of a delay fault on every single path $\pi_i \in \Pi$ for the same direction of transition (rising or falling) at the destination. The terms path and fault will refer to multipath and MPDF, respectively, unless otherwise specified.

Let F be an MPDF on a multipath Π, and let F' be a fault on $\Pi' \subseteq \Pi$ for the same direction of transition as F. Then F' will be called a *subfault* of F, and denoted $F' \subseteq F$. If $F' \subset F$, then F is a proper subfault of F.

A multipath Π is *statically sensitized* by an input vector v if it produces a non-controlling value on every side input of the multipath. A vector *functionally sensitizes* a multipath if it produces non-controlling values on the side inputs of gates whose on-path inputs have the non-controlling value.

A vector-pair $< v_1, v_2 >$ is a *non-robust test* for an MPDF on Π, if it produces an appropriate transition(s) at the source of the multipath, and v_2 sets all side inputs to non-controlling values.

A vector pair $< v_1, v_2 >$ is a *robust test* [2] for a path delay fault F if the output of the circuit in the presence of F is different from that of the fault-free circuit, independent of delays in the rest of the circuit. A robust test must satisfy the above conditions for non-robust tests, and for every gate with the controlling value on on-path inputs, the side inputs must remain steady at the non-controlling value.

3 Primitive faults in combinational circuits

Faults that must be tested to guarantee timing correctness of a circuit are called primitive faults. Such faults were defined in [9] for combinational circuits in terms of sensitizability of paths. We shall use a functional definition instead, so that it applies to both combinational and sequential circuits.

Definition 1: An MPDF F is *primitive* if there exists a vector-pair for which the circuit with *only* the fault F present *may* produce an observable output different from that produced by the fault-free circuit, and there is no $F' \subset F$ with the same property.

The following lemma follows directly from [9, 10].

Lemma 1: An MPDF F in a combinational circuit is primitive if and only if it is non-robustly testable and no subfault of F is nonrobustly testable.

Our method of identifying primitive faults uses the concept of sensitizing cubes [14]. A *cube* is defined as a subset of input literals. A cube can be represented by the values assigned to the inputs, or as a product of literals. Thus, a cube corresponds to a set input vectors, each of which will be referred to as a *vertex*. The vertices in a cube are said to be covered by the cube. A vertex in a cube that is not covered by any other cube is called an *essential vertex*. Each primary output of the circuit will be treated separately.

Definition 2: A minimal set of input values necessary to produce a specific output value (often called mandatory assignments) is called a *sensitizing cube* of the output. There are two sets of sensitizing cubes associated with each output of a circuit: sensitizing 0-cubes and sensitizing 1-cubes.

Definition 3: A path Π is *associated* with a sensitizing cube q if (1) it sets every side-input to the non-controlling value when the on-path input is non-controlling and (2) no side input has a controlling value when the on-path inputs are controlling. We shall refer to q as a sensitizing cube of Π.

Sensitizing cubes can be determined by tracing back from the output and assigning signal values as follows: If a controlling value is necessary to produce the desired gate output, assign the controlling value to a gate input that has not been chosen before. If non-controlling values are required, assign it to all gate inputs. Repeat the process until primary inputs are reached. Justify all assigned line values, making only necessary assignments, and determine all implications. A conflict-free assignment of input variables obtained in this manner corresponds to a sensitizing cube.

For each sensitizing cube, the associated paths are identified using using Definition 3 as follows: For gates with one or more inputs with the controlling value, all gate inputs with the controlling value are on-path inputs of the same multipath. For gates with only non-controlling value inputs, only one of the inputs is selected for the path. All sensitizing cubes and the associated multipaths can be

found by a depth-first traversal of the circuit using the above method. The following example demonstrates the procedure.

Example 1: Consider the circuit (from [14]) and the set of signal values shown in Fig. 1(a). This set of necessary signal values to make $f = 1$ is obtained by selecting the first (upper) input of each gate for assigning the controlling value. The input assignment corresponds to the sensitizing 1-cube

$q_1 = \overline{a}\overline{b} = 0\ 0\ x$

associated with paths $a357f$, $b257f$ and $\{a1457f, b1457f\}$. Similarly, assigning 1 to the lower input of gate 7, we get

$q_2 = \overline{c} = x\ x\ 0$

which sensitizes path $c67f$ to 1. The sensitizing cubes and the paths associated with them, obtained by justifying $f = 0$ are:

$1\ x\ 1$: $a357f$ and $c67f$.

$x\ 1\ 1$: $b257f$ and $c67f$.

Fig. 1(b) shows the value assignments that produce the sensitizing 0-cube $1\ x\ 1$. The cube $x\ 1\ 1$ is obtained by assigning 0 to the third input of gate 5. Note that assigning 0 to the middle input of gate 5 results in a conflict and does not produce a sensitizing cube. The example shows that a sensitizing cube may be associated with more than one multipath. A multipath may also have more than one sensitizing cube for the same output value. □

The above method of computing sensitizing cubes eliminates the need for obtaining collapsed form expressions of the outputs as in [14]. Although the worst case time complexity of the two methods may be the same, the memory requirements of the new method are expected to be lower. Unlike the earlier method, cubes that cannot be sensitizing cubes are not generated at all, resulting in some speed-up. We expect the proposed method to be able to handle larger circuits than the method in [14].

For the sake of convenience, we shall not explicitly specify the direction of transition for faults. A fault on a multipath Π will refer to the MPDF on Π for a rising or falling transition at the destination, depending on whether we are considering sensitizing 1-cubes or 0-cubes. We shall say that the path Π is primitive when the fault on the path with the implied direction of transition is primitive.

The following lemmas from [14] are used in identifying primitive faults in combinational circuits.

Lemma 2: Every essential vertex of a sensitizing cube statically sensitizes a primitive fault.

Lemma 3: Let v be a common vertex of a set $Q = \{q_1, q_2, ..., q_k\}$ of sensitizing cubes, where $k > 1$. The multipath Π sensitized by v is primitive, if and only if (1) no $q_i \in Q$ has an essential vertex, (2) no proper subset of Q has a common vertex and (3) no proper subset of Π is static sensitizable.

Primitive faults in combinational circuits are identified by finding MPDF's that are statically sensitized by vertices satisfying Lemmas 2 and 3. Since a path may have a number of sensitizing cubes, some faults may be identified as primitive more than once. There may also be faults F_i, F_j that satisfy condition (1) of Lemma 3. If $F_i \subseteq F_j$, F_j must be deleted from the set of primitive It has been shown in [22] that all primitive faults in combinational circuits can be identified using Lemmas 2 and 3.

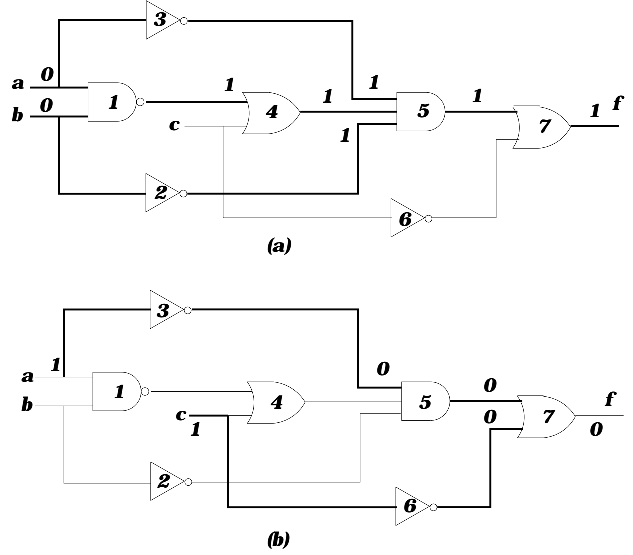

Figure 1. Finding sensitizing cubes.

4 Primitive faults in sequential circuits

In this section, we first present some properties of primitive faults in sequential circuits, and use then to develop an algorithm for identifying them. Proofs of lemmas have been omitted, but can be found in [22].

4.1 Properties

In non-scan sequential circuits, tests can be applied only at primary inputs, and the test results observed at primary outputs. A vector-pair $< v_1, v_2 >$ is necessary to generate an appropriate transition at the source of the path being tested and propagate the transition along the path. Each vector v_i consists of a state component S_i and a primary input component I_i. Since the state variables cannot be controlled directly, the vector-pair must also satisfy the condition that S_2 is reached by applying I_1 to the circuit in state S_1. Such vector-pairs are called *applicable* vector-pairs.

In extending the concept of primitive faults to sequential circuits, we must consider a fault as primitive only if its presence can affect circuit behavior in the normal mode of operation. Fault activation must be with transitions that can be produced in the circuit, and fault effects reaching flip-flops must propagate to primary outputs under the normal clock speed.

In this paper, we consider only MPDF's such that the faulty paths have a common destination, as defined in Section 2. Later, we show why multiple-faults on multipaths to different destinations may have to be considered in some cases.

Lemma 4: An MPDF F on a multipath Π to a primary output is primitive if and only if there exists a vector-pair $< v_1, v_2 >$ such that

(1) it is applicable;

(2) launches an appropriate transition at the source of Π;

(3) v_2 statically sensitizes Π; and

(4) there is no subfault of F which satisfies the above conditions.

When the destination of the faulty path is a state variable flip-flop, a vector-pair that satisfies conditions (1)-(3) above will be said to *activate* the fault.

Lemma 5: An MPDF F on a multipath Π to a state variable flip-flop y_i is primitive if and only if

(1) there exists a vector-pair $< v_1, v_2 >$ that activates the fault;

(2) there exists an input sequence $\underline{I} = I_3, I_4, ..., I_n$ which makes a primary output of the fault-free circuit sensitive to the value at y_i without activating the path Π; and

(3) there is no $F' \subset F$ which satisfies the above conditions.

Since Lemma 5 stipulates that $\underline{I}$ not activate Π, fault propagating sequences for determining whether a fault is primitive can be derived without considering fault effects. This can be done by finding an input sequence that propagates a D from the destination of Π to a primary output, using D-algorithm and time frame expansion [23].

The method presented in Section 3 can be used to find primitive faults in the next-state and output logic, treated as combinational circuits. Primary inputs and flip-flop outputs are treated as inputs, while primary outputs and flip-flop inputs are considered outputs. Faults that are primitive in these circuits will be called *combinationally primitive*. A combinationally primitive fault is primitive in the sequential circuit if a vector-pair that sensitizes it is applicable, and if the path is to a flip-flop, a propagating sequence also exists. The following lemma shows that if a combinationally primitive fault does not satisfy these conditions, certain other faults must be analyzed to determine whether they are sequentially primitive.

Lemma 6: Let vertex v in a sensitizing cube statically sensitize a MPDF F. If an applicable vector-pair $< v_0, v >$ does not exist, or if the fault effect cannot be propagated to a primary output, then a fault $F' \supset F$ may be sequentially primitive.

Before discussing the detailed algorithm for primitive fault identification, we briefly consider the case of faults on paths to two or more flip-flops, alluded to earlier. A vector that statically sensitizes an MPDF to a state variable y_i may also sensitize paths to other state variables. According to Definition 1, a fault occurring alone, must affect an output to be considered primitive. Consider the case where an MPDF F_1 on a path to y_i cannot be propagated from y_i. Assume that the vector-pair that activates F_1 also activates an MPDF F_2 to y_j and that the simultaneous changes at y_i and y_j can be propagated to an output. If F_2 cannot be activated and propagated to an output when occurring alone, the two faults together must be treated as primitive by Definition 1. We shall refer to such a set of MPDF's with different destinations as a *complex MPDF*. The method presented in this paper can be extended to complex MPDF's by considering sets of MPDF's, but may not be practical. While the existence of primitive faults of this type is theoretically possible, we believe that they are unlikely to occur in practice.

4.2 Primitive fault identification

Our primitive fault identification method first derives all sensitizing 0- and 1-cubes for each state variable and each output. Combinationally primitive faults are identified using Lemmas 2 and 3. Essential vertices in the sensitizing cubes of the destination are processed first. For every essential vertex v, we attempt to derive an applicable vector-pair with v as the second vector, using a procedure that will be described later. If the destination is a flip-flop input, we also try to derive an input sequence that propagates the fault effect to a primary output. If successful, the fault is put in a list of potential primitive faults, provided it is not already in the list. Otherwise, the vertex is flagged.

Next, vertices common to two or more cubes satisfying Lemma 3 are processed. Let v be a common vertex of a set of cubes Q satisfying Lemma 3. If an applicable vector-pair and a propagating sequence are found, the fault is compared with those in the list of potential primitive faults and added to the list, ensuring that the list does not contain duplicates or faults that contain other faults in the list. Otherwise, the common vertex is flagged, and the process is repeated for all common vertices v', such that v' is a vertex in some cube in Q and satisfies Lemma 3. The flagged vertices are ignored during this step. The procedure *Applicable*, shown in Fig. 2, uses sensitizing cubes to find applicable vector-pairs. For a given vector v, it a returns a vector v_1 such that $< v_1, v >$ is applicable. Each call returns a new vector, and if no suitable vector exists a null vector Φ is returned. Procedure *Sequential_primitive* given in Fig. 3 identifies all primitive faults in a sequential circuit. It uses the recursive procedure *Common_vertex*, shown in Fig. 4, to process common vertices.

Example 2: Consider the part of the output logic of a sequential circuit shown in Fig. 5. The sensitizing 1-cubes of the circuit are shown in Fig. 5(c). The combinationally primitive faults are sensitized by the essential vertices 001, 111, 100 and by the common

```
PROCEDURE Applicable(v)
BEGIN
Returns a vector v_i such that < v_i, v > is applicable;
Φ if impossible
(y_1, y_2, ...y_n): state component of v
Q_i, 1 = 1, 2, ..., n: sensitizing cubes
C_i^0, C_i^1: Sensitizing cubes of Y_i variable

  FOR i = 1 through n
    IF (y_i = 0)
      Q_i = C_i^0
    ELSE
      Q_i = C_i^1

  Compute P = {p_1, p_2, ...p_k}, where each p_i
  is the intersection of cubes,one from each set Q_i

  FOR each untried vector v_i in P
    IF < v_i, v > creates appropriate transitions
        at start of path
      Mark v_i as tried
      return v_i
    ELSE
      Mark v_i as tried
  return Φ
END
```

Figure 2. Finding an applicable vector-pair.

PROCEDURE Sequential_primitive
S_0, S_1: set of sensitizing cubes for a PO or FF input
L_0, L_1: rising and falling transition primitive faults
BEGIN
  FOR $i = 0, 1$
    $L_i = \Phi$
    FOR every essential vertex v
      f = fault statically sensitized by v
      IF Applicable(v) = Φ
        Flag v
      ELSE IF destination = FF
        IF PO cannot be sensitized
          Flag v
      IF v is not flagged
      $L_i = L_i \bigcup f$
    FOR every common vertex v satisfying Lemma 3
        Common_vertex(v, Q, i)
END

Figure 3. Primitive fault identification.

PROCEDURE Common_vertex(v, Q, i)
BEGIN
  f = fault statically sensitized by v
  IF Applicable(v) = Φ
    Flag v
  IF destination is an FF
    IF PO cannot be sensitized
      Flag v
  IF v is flagged
    FOR every $v' \in q'$; $v' \neq v$; $q' \in Q$;
      IF v' is a common vertex of Q' and satisfies Lemma 3
        Common_vertex(v', Q', i)
  ELSE   FOR all $f' \in L_i$
      IF $f \subset f'$
        $L_i = L_i - f'$
    IF $f' \not\supseteq f$ for any $f' \in L_i$
      $L = L_i \bigcup f$
END

Figure 4. Static sensitizing vector identification.

vertex 010. Assume that appropriate transitions into all of them except 111 are available in the sequential circuit. Since the path sensitized by 111 is not sequentially primitive, it is flagged. Ignoring the flagged vertex, i.e., treating it as a non-essential vertex, vertex 110 satisfies Lemma 3. The multipath shown in Fig. 5(b) sensitized by the common vertex 110 will be primitive, if an appropriate transition to it exists. Note that the single path shown in Fig. 5(a) sensitized by 111 is contained in the multipath sensitized by 110. □

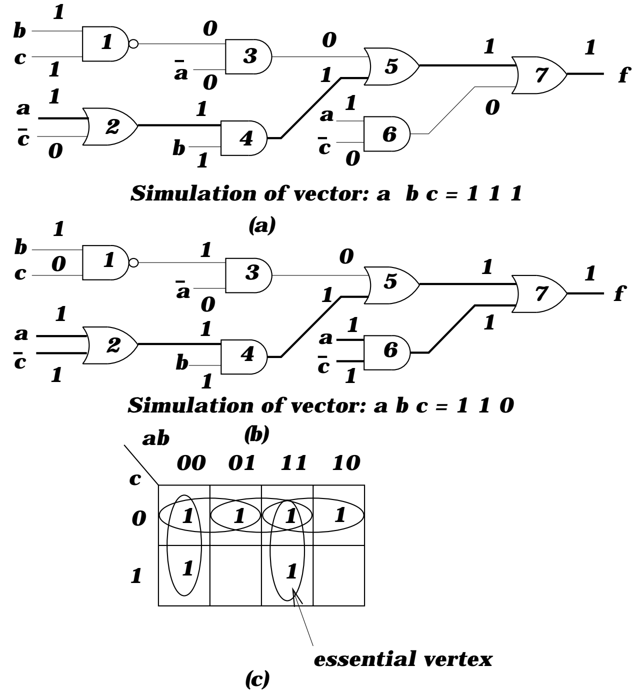

Figure 5. Circuit and sensitizing cubes of Example 2.

5 Test Generation

The test generation algorithm assumes that the circuit to be tested has a reset input which takes it to a unique reset state which is not affected by any fault. A test for a path delay fault consists of three parts: initialization, fault activation, and propagation to a primary output. The last part is required only for faults on paths that terminate at flip-flops. The initializing and propagating sequences will be applied with a "slow clock" whose period is such that delay faults have no effect [24].

5.1 Fault activation

The procedure for identifying primitive faults (Fig. 3) produces an applicable vector-pair $< v_1, v_2 >$, such that v_2 statically sensitizes each fault. This vector-pair is checked to see whether it satisfies the robustness condition. If not, procedure *Applicable* is used

repeatedly to find another initial vector v_1', such $< v_1', v_2 >$ satisfies the robustness condition, or no new applicable $< v_1', v_2 >$ remains.

5.2 Initializing sequence

Once an applicable vector pair $< v_1, v_2 >$ is obtained, an input sequence that takes the circuit from the reset state to S_1 is obtained. Let $S_1 = (y_1, y_2, ..., y_n)$ and let $Q = (Q_1, Q_2, ..., Q_n)$, where Q_i is the set of sensitizing 1(0)-cubes if $y_i = 1(0)$. The state component of each intersection of cubes, one from each set Q_i, represents a state (or a set of states if some variables are unspecified) from which S_1 can be reached. The input part is the input needed for that transition. If any intersection contains the required reset state, the input part of the intersection will take the circuit from the reset state to S_1. Otherwise, for each intersection, we treat the state component as the new state to be reached, and repeat the procedure until the reset state appears in one of the intersections. The input components of the intersections along the path to the reset state constitute the initializing sequence. Note that the method finds transitions only as needed.

5.3 Fault propagation to primary outputs

Delay fault effects that reach state variable flip-flops may produce incorrect next-state values at one or more flip-flops. These values can be propagated to primary outputs as in stuck-at fault testing, since delay faults will not have any effect due to the "slow clock" used during this phase. If the fault effect reaches only a single state variable, a sequence for propagating it to a primary output can be derived using D-propagation and the iterative array model of the sequential circuit.

A vector pair that robustly activates a path terminating at a particular state variable may also sensitize paths to one or more other state variables, not necessarily robustly. Thus, the state reached by the circuit may depend on delays on paths other than the one tested. If k state variables may be affected, including the destination of the target fault, there are $2^k - 1$ possible incorrect patterns to be distinguished from the correct one. A method using multiple propagating sequences, similar to iterative array testing [23, 25] is possible but may not be practical for large circuits.

We propose an iterative method of fault propagation which treats faults whose effects have been successfully propagated in earlier iterations as not present in the circuit during the current iteration. These faults are treated as having been tested. Initially the set of tested faults is empty. In each iteration, all flip-flops whose values could be affected during each test, except the destination of the path being tested and those in the tested set, are assigned the unknown value x. Values at the destinations of tested paths are set their fault-free values. D-propagation from the destination flip-flop to a primary output is attempted with time-frame expansion. If a sequence is found within a specified number of time frames, the fault is added to the set of tested faults. The procedure is repeated until no more faults can be added to the set of tested faults. The procedure is given in Fig. 6.

This method may not be able to derive propagating sequences for all testable faults, but the sequences generated will be correct. The tests derived by this method are similar to validatable non-robust tests, since the propagation of a fault from a flip-flop to a primary output is valid if certain other faults are tested and proven to be absent.

PROCEDURE Propagate
BEGIN
  $A = \Phi$
  DO
    $B = A$
    FOR every unpropagated test $< v_1, v_2 >$
      F = set of faults robustly activated
          by $< v_1, v_2 >$
      y_i = destination of faults in F
      Set $y_i = D$
      FOR every y_j, $j \neq i$ that is affected by F
        IF y_j is not the destination of a fault in A
          Set $y_j = x$
        ELSE
          Set $y_j = Y_j(v_2)$
          (fault-free value produced by v_2)
      Set remaining y_k's to $Y_k(v_2)$
      IF D-propagation to PO is successful
        $A = A \bigcup F$
  WHILE $A \neq B$
END

Figure 6. Procedure for fault propagation.

6 Experimental Results

A program implementing the primitive fault identification and test generation methods proposed in this paper has been implemented in the C language. Primitive faults were identified and robust tests were derived for a number of sequential circuits in ISCAS'89 and MCNC'91 benchmarks. Experimental results are summarized in Table 1. As mentioned in Section 5, our experiments assumed that the fault-free and faulty circuits are started in a reset state which is unaffected by the fault. This implies that faults that affect initialization will not be present in the circuit and are excluded in the results reported. This restriction can be easily removed.

The first two columns give the circuit name and the number of gates in the circuits considered. Faults are divided into two groups: those on paths to primary outputs, and those on paths that end on flip-flops. Experimental results are also grouped accordingly in Table 1, with a third group including all faults. Within each group, we give the total number of single faults, the total number of primitive faults identified and the number of primitive faults for which robust tests were obtained. Our algorithm identifies all primitive faults on paths to primary outputs but, not all those on paths to flip-flops. Therefore, upper bounds on primitive faults on paths to flip-flops are also given in the table. Times given in the last column are total CPU times for primitive fault identification and test generation, on an IBM RS 6000 server.

Our algorithm considers all applicable vector-pairs that satisfy the conditions for sensitization during primitive fault identification.

Similarly, all appropriate vector-pairs are considered until one that robustly sensitizes each primitive fault is found. However, only a limited search is used in propagating fault effects from flip-flops to primary outputs. For determining whether a fault on a path to a flip-flop is primitive, we first try to find an input vector that propagates it to a primary output. If successful, the fault is primitive. Otherwise, for each correct/incorrect state pair reached at the end of the first clock cycle, we try propagate the fault effect to a primary output with sequence of length ≤ 4. If propagation to a primary output is unsuccessful, the fault may or may not be primitive. These faults account for the difference between the upper bounds and the primitive faults given in the table. A similar approximation was also used in fault propagation during test generation. Our results indicate there were many faults for which we could not determine whether they were primitive. The coverage of primitive faults was also low in all the circuits in our experiments. Clearly, the coverage of primitive faults is somewhat higher, even if the number of primitive faults equals the upper bound given in the table.

Both the primitive fault identification and test generation results can be improved by using more sophisticated fault propagation techniques. Since fault propagation is done in the gate-level circuit, the use of controllability and observability measures should also give better results. In the case of test generation, our algorithm assigns the unknown value (x) to flip-flops unless the path to it has been tested robustly. This leads to pessimistic results. The method can be improved by considering all combinations explicitly, when when only a few flip-flops are affected by the fault.

7 Conclusion

We have extended the concept of primitive faults, previously defined only for combinational circuits, to sequential circuits. By using a functional definition, the same definition applies to both combinational and sequential circuits. Problems like faults that prevent initialization, and start-up states that cannot be reached from other states are handled in a natural way.

We have presented an algorithm which, in principle, can identify all primitive faults in sequential circuits and generate robust tests for all robustly testable primitive faults. It uses the concept of sensitizing cubes introduced in an earlier paper, and uses a more efficient algorithm to generate the cubes. These sensitizing cubes are used for obtaining static sensitizing vectors for identifying primitive faults as well as deriving robust tests for them. We have introduced a new method for fault propagation which gaurantees that the tests produced are validatable non-robust tests.

The proposed method of primitive fault identification identifies only primitive MPDF's, which by definition have a common destination. We have shown that it is possible for a primitive fault to contain paths to more than one flip-flop. A new class of faults called complex MPDF's consisting of MPDF's to two or more destinations must be considered for identifying all primitive faults. Extensions of the proposed method to complex MPDF's is under investigation.

The effectiveness of our method of primitive fault identification and test generation has been demonstrated for sequential circuits of moderate size. The cube-oriented operations may make the method unsuitable for very large circuits. An important area for further investigation is the development of methods that operate entirely on the circuit model. Another area for further work is on algorithms for fault effect propagation. The use of testability measures is likely to extend the usefulness of the techniques developed in this paper.

References

[1] G.L. Smith, "Model for delay faults based upon paths", *Proc. Int. Test Conf.,* pp. 342-349, Nov. 1985.

[2] C.J. Lin and S.M. Reddy, "On delay fault testing in logic circuits," *IEEE Trans. on CAD*, pp. 694-703, Sept. 1987.

[3] C. Lin, S.M. Reddy and S. Patil, "An automatic test pattern generator for the detection of path delay faults,", *Proc. Int. Conf. on Computer Aided Design*, pp. 284-287, Nov. 1987.

[4] M.H. Schultz, K. Fuchs and F. Fink, "Advanced Automatic Test pattern Generation techniques for path delay faults," *Proc. Int. Symp. on Fault-Tolerant Computing*," pp. 445-51, June 1989.

[5] S. Devadas and K. Keutzer, "Validatable non-robust delay-fault-testable circuits via logic synthesis," *IEEE Trans. on CAD*, vol. 11, pp. 1559-1573, Dec. 1992.

[6] K.-T. Cheng and H.C. Chen, "Delay testing for non-robust untestable circuits," *Proc. Int. Test Conf.*, pp. 954-971, Oct. 1993.

[7] I. Pomeranz, S.M. Reddy and P. Uppalpuri, "NEST: A non-enumerative test generation method for path delay faults in combinational circuits," *IEEE Trans. on CAD*, vol. 14, pp. 1505-1515, Dec. 1995.

[8] W.K. Lam, A. Saldanha, R.K. Brayton and A.L. Sangiovanni-Vincentelli, "Delay fault coverage and performance tradeoffs," *Proc. Design Automation Conf.*, pp. 446-452, June 1993.

[9] W. Ke and P.R. Menon, "Delay-Verifiability of Combinational Circuits based on Primitive Faults," *Proc. Int. Conf. on Computer Design*, pp. 86-90, Oct. 1994.

[10] W. Ke and P.R. Menon, "Synthesis of delay-verifiable combinational circuits", *IEEE Trans. on Computers.*, vol. 44, pp. 213-222, Feb. 1995.

[11] D.B. Armstrong, "On finding a nearly minimal set of fault detection tests for combinational logic nets," *IEEE Trans. Electron. Computers*, vol. EC-15, pp. 66-73, Feb. 1966.

[12] M. Sivaraman and A. J. Strojwas, "Primitive path delay fault identification," *Proc. Int. Conf. on VLSI Design*, pp. 95-100, Jan. 1996.

[13] A. Krstic, K.-T. Cheng and S.T. Chakradhar, "Identification and test generation for primitive faults," *Proc. Int. Test Conf.*, pp. 423-432, Nov. 1996.

[14] R. Tekumalla and P.R. Menon, "Test generation for primitive path delay faults in combinational circuits," pp. 636-641, *Proc. Int. Conf. on Computer Aided Design*, Nov. 1997.

Circuit	Gates	Faults to PO's			Faults to FF's				Aggregate			Time
		Total	Primitive	Robust	Total	Primitive		Robust	Total	Primitive	Robust	*Sec*
						UB[a]	Actual					
s208	104	108	81	54	176	176	75	39	284	156	93	10.68
s298	119	12	14	12	450	328	205	64	462	219	76	22.41
s344	160	34	25	25	676	608	527	160	710	552	185	102.03
s349	161	34	25	25	696	609	527	160	730	552	185	91.31
s382	158	12	12	12	788	655	201	46	800	213	58	199.45
s386	159	182	175	83	232	207	182	82	414	357	165	42.32
s400	162	12	12	12	884	670	202	46	896	214	58	223.09
s420	218	340	219	170	608	608	179	115	948	398	285	222.29
s444	181	12	12	12	1058	695	188	46	1070	200	58	145.47
s510	211	218	197	80	520	486	361	119	738	558	199	99.66
s526	193	12	12	12	808	670	221	67	820	233	79	113.39
s641	379	1014	618	236	2430	1338	961	311	3444	1579	547	1235.41
s713	393	11226	1260	273	27598	2763	2128	529	43624	3388	802	1578.32
s820	289	324	317	144	660	597	508	109	984	825	253	336.81
s832	287	324	306	141	688	559	475	88	1012	781	229	323.36
s1488	590	1918	1356	374	844	667	613	341	2762	1969	715	1966.08
s1494	472	1206	1048	359	858	773	675	349	2064	1723	708	2068.81
s1196	529	4944	2152	926	1250	785	296	137	6194	2448	1063	1142.51
s1238	508	5778	2998	1107	1338	804	309	143	7116	3307	1250	2750.62
bbsse	101	146	133	68	212	196	175	75	358	308	143	13.89
cse	146	330	220	102	584	462	420	80	914	640	182	58.94
dk16	244	144	124	34	754	520	394	85	898	518	119	152.95
keyb	2238	124	88	28	1068	795	476	57	1192	564	85	275.20
planet	646	2092	1471	377	1934	1307	1193	108	4026	2664	405	2287.07
sse	117	168	144	74	242	189	166	74	410	333	148	19.24
scf	916	2762	1884	607	4274	3084	2090	361	7036	3974	968	2070.01
s1	446	944	642	167	1558	1044	991	109	2502	1633	276	2217.77
styr	477	862	686	190	1102	718	699	203	1964	1385	393	2278.60
tma	140	288	244	73	218	205	179	69	506	423	142	36.76

[a] Upper Bound

Table 1. Primitive Fault Coverages for ISCAS'89 and MCNC'91 benchmark circuits.

[15] R. Tekumalla and P.R. Menon,"Delay Testing with Clock Control: An Alternative to Enhanced Scan," *Proc. Int. Test Conf.*, pp. 454-462, Nov. 1997.

[16] K.-T. Cheng, S. Devadas and K. Keutzer, "A partial enhanced-scan approach to robust delay-fault test generation for sequential circuits," *Proc. Int. Test Conf.,* pp. 403-410, Oct. 1991.

[17] R. Tekumalla and P.R. Menon, "Synthesis of delay verifiable sequential circuits using partial enhanced scan," *Proc. Int. Conf. on Computer Design*, pp. 648-653, Oct. 1997.

[18] W. Ke and P.R. Menon, "Path Delay fault testable non-scan sequential circuits," *Proc. IEEE Trans. on CAD*, vol.14, pp. 576-582, May 1995.

[19] R. Tekumalla and P.R. Menon, "Delay testable non-scan sequential circuits with clock suppression," *Proc. Int. Symposium on Circuits and Systems.,* pp. 137-140, May 1996.

[20] I. Pomeranz and S.M. Reddy, "On identifying undetectable and redundant path delay faults in synchronous sequential circuits," *Proc. 12th IEEE VLSI Test Symposium*, pp. 8-14, April 1994.

[21] R. Tekumalla and P.R. Menon, "Identifying redundant path delay faults in sequential circuits," *Proc. Int. Conf. on VLSI Design*, pp. 406-411, Jan. 1996

[22] R. Tekumalla and P.R. Menon, "Primitive faults in combinational and sequential circuits," *Tech. Rept.*, TR-CSE-98-7, Univ. of Massachusetts, Amherst, August 1998.

[23] M. Abramovici, M.A. Breuer and A.D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1990.

[24] Y.K. Malaiya and R. Narayanaswamy, "Testing for timing faults in synchronous sequential circuits," *Proc. Int. Test Conf.*, pp. 560-571, Oct. 1983.

[25] A.D. Friedman, "Easily testable iterative systems," *IEEE Trans. on Computers*, vol. C-22, pp. 1061-1064, Dec. 1973.