2001

Design Verification and Functional Testing of Finite State Machines

Mark W. Weiss
University of Nebraska-Lincoln, mweiss@cse.unl.edu

Sharad C. Seth
University of Nebraska-Lincoln, seth@cse.unl.edu

Shashank K. Mehta
Indian Institute of Technology, Kanpur, skmehta@cse.iitk.ac.in

Kent L. Einspahr
Concordia University, Seward, NE, Kent.Einspahr@cune.edu

Fourteenth International Conference on VLSI Design, 2001. doi: 10.1109/ICVD.2001.902659

Design Verification and Functional Testing of Finite State Machines

Mark W. Weiss and *Sharad C. Seth*
University of Nebraska-Lincoln
Lincoln, NE 68588-0115 USA
{mweiss,seth}@cse.unl.edu

Shashank K. Mehta
Pune University
Pune, 411007 India
skm@cs.unipune.ernet.in

Kent L. Einspahr
Concordia University
Seward, NE 68434 USA
eins@seward.cune.edu

Abstract

The design of a finite state machine can be verified by simulating all its state transitions. Typically, state transitions involve many don't care inputs that must be fully expanded for an exhaustive functional verification. However, by exploiting the knowledge about the design structure it is shown that only a few vectors from the fully expanded set suffice for both design verification and testing for manufacturing defects. The main contributions of the paper include a unified fault model for design errors and manufacturing faults and a function-based analysis of the circuit structure for the purpose of generating tests under the unified model. Experimental results on benchmark finite state machines are presented in support of this approach to test generation.

1. Introduction

Test generation for design verification and manufacturing faults are generally regarded as independent activities. At times, manufacturing tests may be augmented with design verification test vectors to catch "unmodeled" faults although a sound basis for combining the two kinds of tests does not exist. In this paper we propose a unified approach to testing and verification of finite state machines (FSMs).

We assume that the FSM design is verified by simulating its state transitions. Typically, a state transition involves many don't care inputs which must be fully expanded for an exhaustive functional verification. Alternatively, an analysis of the circuit structure in the context of the specific state transition allows us to select only a subset of the fully expanded vectors without losing any coverage of faults under the unified model. These vectors are simulated individually for design verification and are included in a tour of the FSM states to define a manufacturing test. Thus, both kinds of tests are derived from a common basis.

Test generation using the functional description of a FSM, with or without the circuit implementation, is not new. Purely function-based test generation methods have used the single-transition fault model [1] and its extension to multiple state-table faults [2]. However, test sequences based solely on the functional information tend to be long because they must work for *any* implementation. Further, the generated test must be simulated on the specific implementation to determine its fault coverage.

As in other prior works [3], [4], we assume that the test generator can access the gate-level implementation. However, while we consider design verification and manufacturing testing in a unified fashion, the earlier work focuses only on manufacturing testing.

The rest of the paper is organized as follows: Section 2 describes a unified fault model for design errors and manufacturing defects. Next, in Section 3, a test generation technique is discussed for design verification and functional testing for manufacturing defects. The implementation of this method involves exact three-value simulation, fault list computation, and constrained test generation, as explained in Section 4. The results of the evaluation of tests generated using border-gate analysis are presented in Section 5. Section 6 concludes the paper.

2. A Unified Fault Model

Manufacturing faults (such as stuck-at and bridging) and design errors (such as wrong-gate-substitution, missing-gate, extra-input, missing input, etc.) can be unified into a single model. Let $\mathcal{G}$ be a "good" circuit, i.e., it conforms to its specifications. The faulty circuits are described by the pair $\mathcal{G}, \mathcal{F}$, where $\mathcal{F}$ is the *fault list*. $\mathcal{F}$ is defined by the set of pairs, $\{(S_1, E_{S_1}), (S_2, E_{S_2}), \ldots, (S_k, E_{S_k})\}$, where each S_i is a collection of lines of the circuit $\mathcal{G}$ and E_{S_i} is the corresponding *environment* condition. In the interpretation of the fault (S_i, E_{S_i}), if any of the E_{S_i} conditions are satisfied, then all lines of S_i in $\mathcal{G}'$ have complementary values compared to their respective values in $\mathcal{G}$.

Example faults described in the unified model include the following. The stuck-at-1 fault at line g is given by $(\{g\}, \{g = 0\})$. A bridging fault in which line b at 1 forces line a to 1 is expressed as $(\{a\}, \{ab = 01\})$. A gate substitution error at a gate output g, in which a two-input AND gate is replaced

by a two-input OR gate, can be captured by the fault $(\{g\}, \{h_1h_2 = 01\ OR\ h_1h_2 = 10\})$, where, $h1$ and $h2$ are inputs to the gate.

The unified model can be used to generate a test for manufacturing faults in the same way as for the stuck-at or the bridging fault model. A test sequence for fault (S, E_S) must excite the fault by satisfying the condition E_S and then must propagate the faulty signal from one of the lines of S to a PO.

The given circuit is assumed to be correct for the purpose of generating manufacturing tests; each fault of the model is considered in conjunction with the correct design for coverage by a test. The same approach cannot be taken for design verification because the circuit to be verified is possibly an incorrect implementation of the FSM specification. Nevertheless, we generate the tests for design verification in the same manner as for manufacturing because most design errors of the model are reversible. For example, if the bad circuit results from the good one by 'X-gate substituted by Y-gate' then the good circuit results from the bad by 'Y-gate substituted by X-gate'. Other pairs of complementary errors are extra-gate & missing-gate, and extra-input & missing-input. This approach allows us to generate a test from the bad circuit using the model which can distinguish from the variant, namely, the good circuit. Once again, test generation for (S, E_S) involves creating excitation and propagation conditions.

3. Test Generation

3.1 Design Verification Tests

An implementation of a finite state machine (FSM) is a sequential circuit, but its verification is equivalent to the verification of the underlying combinational circuit since the designer can control the secondary inputs and observe the secondary outputs. Let the collection of the transitions of the FSM be $R = \{(\langle S_i, I_i\rangle/\langle T_i, O_i\rangle)|i \in \mathcal{I}\}$, where S_i is the initial state, T_i is the final state, I_i is the primary input, and O_i is the primary output of the i-th transition. In the underlying combinational circuit, the secondary inputs and secondary outputs are treated as additional PIs and POs, respectively. Therefore, its specification is $\{(\langle I_i'\rangle/\langle O_i'\rangle)|i \in \mathcal{I}\}$, where I_i' is composed of the bits of I_i and the bits of the encoding of S_i, and O_i' is composed of the bits of O_i and the bits of the encoding of T_i. The design is correct if and only if it performs each transition correctly. That, in turn, is equivalent to verifying the correctness of each input/output pair $(I_i'/O_i')\forall i \in \mathcal{I}$.

An important point to note is that if there are don't care values in I_i' they must either be simulated symbolically or expanded fully. Computationally, both options can be very expensive. We propose an alternative below.

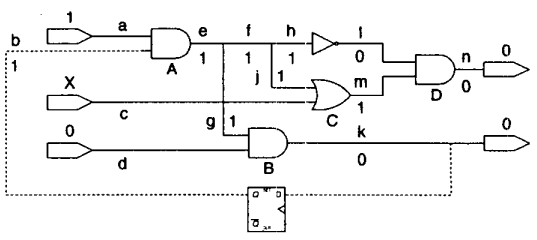

Fig. 1. Illustration of $\mathcal{F}$ classes.

Consider a modulo-8 counter. The underlying combinational circuit has a 3-bit input and a 3-bit output. Testing the complete functionality of the circuit requires setting the input to each of the 8 patterns and comparing the output to the corresponding specified pattern. Such a specification leaves no choice to the functional test-generator to improve the speed of testing since all input patterns have distinct output patterns. Fortunately, in most large circuits the output patterns are much fewer than the valid input patterns. This enables us to specify the functionality of the circuit by forming cubes in the input space and assigning one output pattern to each cube.

When there are input don't cares, a test-generator can optimize the test set by selecting a subset of the vectors of each cube with the same fault testability as the entire cube. For example, in the priority-encoder described above, it may not be necessary to test all eight inputs embedded in XXX1 if say, 1001 and 0011 could test all the faults that could possibly be tested by the vectors of XXX1.

For some partially specified input I_i' we can classify the faults detected by the vectors of I_i' in three classes:

- $\mathcal{F}_{iN}$ the faults that cannot be propagated by any setting of X's,
- $\mathcal{F}_{iA}$ the faults that are propagated by all settings of X's, and
- $\mathcal{F}_{iP}$ the faults that are propagated by some but not all settings of X's.

Sample faults of each class for the circuit in Figure 1 include the following. Faults $(\{l\}, \{lm = 01\})$, and $(\{n, k\}, \{nk = 00\})$ are in $\mathcal{F}_{iA}$. Faults $(\{m\}, \{lm = 01\})$ and $(\{g, d\}, \{gd = 10\})$ are in $\mathcal{F}_{iN}$. Faults $(\{e\}, \{ab = 11\})$ and $(\{a, b\}, \{ab = 11\})$ are in $\mathcal{F}_{iP}$.

The faults of $\mathcal{F}_{iN}$ cannot be detected and any vector of the cube I_i' can test the faults in $\mathcal{F}_{iA}$. Any test for $\mathcal{F}_{iP}$ will also test for faults in $\mathcal{F}_{iA}$. Therefore, for test-generation it is sufficient to consider $\mathcal{F}_{iP}$

faults. Still, the process may not be efficient because $\mathcal{F}_{iP}$ is, in general, a large class. There are very rare instances when a fault set $S = \{l_1, l_2, \ldots, l_m\}$ has a test vector but none of the singleton fault-sets, $\{l_i\}$, are detectable. Therefore, without any significant loss, we only consider a subset of $\mathcal{F}_{iP}$, namely, $\mathcal{F}_{iP}^{singleton}$ which is $\{(S, E_S) \in \mathcal{F}_{iP} | \ S = singleton\}$.

Next, we show that a subset $\mathcal{F}_i$ of $\mathcal{F}_{iP}^{singleton}$ exists with the property that any test set which can detect all faults of $\mathcal{F}_i$ also detects all faults of $\mathcal{F}_{iP}^{singleton}$. To determine the fault set $\mathcal{F}_i$ it is necessary to understand how a vector of cube I_i' performs as a test vector. This is best explained in terms of the results of exact three-value simulation of I_i' on the circuit.

Three-value simulation of a circuit with partially specified input I_i' will be called *exact* when each line is assigned a binary value if and only if it assumes that value for all vectors of I_i'. The problem of computing exact three-value simulation is NP-complete since SAT can be reduced to it. Although the standard three-value simulation is linear in circuit size it is not always exact. An algorithm for exact three-value simulation is presented in the next section.

We define a *border gate* (a gate at the boundary of the X-domain in the simulation) as the gate which has a binary output and at least one X input in an exact three-value simulation. It can be easily verified that the binary output must correspond to the dominating value for the gate (e.g. 0 for AND) in this definition. For input cube $\langle 11X0 \rangle$ in the circuit of Figure 1 the only border gate is C.

Two test vectors of the same cube I_i' differ in their testing capability because they create different conditions at border gates. In Figure 1, vector 1100 allows the faulty signal to pass from j to m at the border gate C. On the other hand, vector 1110 blocks the passage of the faulty signal through C. Using this fact we shall show that there exists a fault subset $\mathcal{F}_i$ of $\mathcal{F}_P$ which is sufficient to consider for test generation.

$\mathcal{F}_i$ can be computed easily from border gate analysis. If a fault $S = \{l\}$ is in $\mathcal{F}_{iP}$, there exists a setting of unspecified PIs which enables the propagation of the fault from l to some PO(s), and there also exists a setting which blocks the propagation. Thus, there must exist a sensitization path starting from l and entering at least one border gate. The sensitization path either (i) passes through no fanout-stem and enters input line l' of a border gate, or (ii) it passes through a fanout-stem and the first such stem is l''.

In case (i) the fault l can be observed only if fault l' can be observed. In case (ii) the fault l can be observed only if the fault l'' can be observed because the sensitization path did not fork before entering l''. This fact leads to the conclusion that it is sufficient to consider faults at the inputs of the border gates and at the fanout-stems in the cone of border gates. The precise class of faults in each category can be determined by classifying border gates as follows:

- *Type-0*: Border gates for which no input has dominating value. Note that in this case the X values on the inputs must be negatively correlated for the output of the gate to be binary.
- *Type-1*: Border gates in which exactly one input has dominating value.
- *Type-2*: Border gates in which two or more inputs have dominating value.

Type-0 border gates can only have inputs with value X or the non-dominating value. For these gates, we need to include only the faults for each X-input line, l, with the environment condition: line l set to the dominating value and all other border gate input lines set to the non-dominating value. These are the only faults that can be propagated through Type-0 border gates.

Similarly, the only input faults that can be propagated through Type-1 border gates involve an input line, l, with the dominating value. The corresponding environment condition is line l set to the dominating value and all other border gate input lines set to the non-dominating value.

No input faults of Type-2 border gates can be propagated because there are multiple dominating inputs. However, border gates of this type can be used to restrict the set of fanout-stem faults described earlier. It can be seen that the only fanout stem faults that are not already covered by the Type-0 and Type-1 border gate faults must be detected by multiple sensitized paths passing through a Type-2 border gate. For such a stem fault to be detected, it must have a binary value and be in the cone of influence of *all* the dominating inputs of the border gate. In summary, we make the following observation regarding the faults in $\mathcal{F}_i$.

Observation

For any input-cube I_i', the singleton faults C_i that cover all faults of $\mathcal{F}_P$ of cube x_i in propagation is the union of the set of X-input lines in Type-0 border gates, the set of dominating input lines in Type-1 border gates, and the set of binary-valued fanout stems, in the cone of all the dominating inputs of Type-2 border gates.

For the circuit of Figure 1, $\mathcal{F}_{\langle 11X0 \rangle}$ is $\{\{j\}\}$.

Once we find $\mathcal{F}_i$ a test set $\mathcal{T}_i$ is computed to propagate the faults of $\mathcal{F}_i$. This test ensures propagation of

```
Tour(Input:V, R;Output:E){
  G = (V, T);
  Find shortest paths P_{α,β} from α to β ∀α, β ∈ V s.t.
       Δ(α) > 0 > Δ(β);
  G' = G;
  While ∃α s.t. Δ(α) > 0 in G' Do
     Select α with Δ(α) > 0 in G';
     Select β from all β' with Δ(β') < 0 s.t. P_{α,β} is shortest;
     Add min{Δ(α), Δ(β)} copies of the edges (transitions)
          of P_{α,β} to G';
  Find an Eulerian cycle E in G'.}
```

Fig. 2. An algorithm to generate a minimum tour of the FSM. $\Delta(x)$ denotes outdeg(x)−indeg(x).

all faults of $\mathcal{F}_P$ and, if $\mathcal{T}_i$ is non-empty, all faults of $\mathcal{F}_A$. Faults $\mathcal{F}_N$ do not have to be considered because they are not detectable by any vector of the cube I_i. If $\mathcal{T}_i$ turns out to be empty (i.e., when $\mathcal{F}_P$ would be empty), then any randomly selected vector of I_i is included in it to take care of $\mathcal{F}_A$. The final test is derived from $\mathcal{T} = \bigcup_i \mathcal{T}_i$ as described in the next subsection.

Finally, we turn to the fault excitation problem. An unrestricted fault model requires us to consider all possible environmental conditions, leading to an unacceptably large test set. Therefore, in our experiments we have considered each fault of $\mathcal{F}_i$ only once for test generation for each input cube. But if the same fault occurs in $\mathcal{F}_i$ and $\mathcal{F}_j$, then the test is generated for it in both of the cases.

3.2 Functional Tests for Manufacturing Faults

Unlike design verification, a functional test for manufacturing defects is more difficult because neither the secondary inputs are controllable nor the secondary outputs observable. Empirically, the effectiveness of the many tour-based functional test methods [3], [5] indicates that distinguishing the faulty state from the good one by an arbitrary vector sequence is not difficult if it is long enough and the FSM is reduced. Therefore, in this work we propose to perform sequential circuit testing by a tour $\mathcal{E}$ of the states of the FSM which covers all the transitions of $\mathcal{T}$ given in the previous section. The tour must not pass through any invalid state otherwise the test will not be functional. The algorithm for the computation for $\mathcal{E}$ appears in Figure 2. Here V denotes the set of states and R is the set of transitions. The algorithm computes $\mathcal{E}$ as the shortest closed walk passing through all the edges of the labeled graph $(V, \mathcal{T})$. An Eulerian cycle (cycle passing through each edge exactly once) exists if and only if the in-degree and the out-degree match for every node. This is achieved in $(V, \mathcal{T})$ by adding copies of some of the edges (making it a multi-labeled graph).

3.3 Related Prior Work

We introduced the border gate approach earlier in the context of combinational logic verification [6]. The key idea of our approach, setting input don't cares to maximize path sensitization in the circuit, is closely related to earlier papers on automatic test pattern generation for manufacturing faults.

RAPS (Random Path Sensitization) [7] and SMART [8] have a similar goal of generating tests that deliberately sensitize a large number of signal paths towards the POs without targeting specific faults. Unlike this paper, however, they assume no primary input constraints.

SMART's restart gates are related to our border gates. A gate is defined to be a *restart gate* if it has one controlling input, its output is critical, and none of its inputs are critical. This can happen only if some of the inputs to the gate are unspecified and the output is specified. Thus, restart gates are border gates but the converse is not true. For example, gate C in Figure 1 is a border gate but not a restart gate because its output is not critical.

The approach presented in this paper is similar to the SMART approach in using border (restart) gates to help extend sensitized paths. The main difference is that SMART ignores multi-branch sensitization paths, which appear more frequently in larger and more complex designs. The multiple branches may pass through the same gate when gates have more than one controlling input so such cases cannot be ignored. Further, treating one restart gate independent of the others cannot handle the sensitized paths with branches in different restart gates.

4. Implementation

The observation in the last section provides the basis for a scheme to generate tests that cover all the faults $\mathcal{F}_P$ for a FSM transition. Recall that for design verification the secondary inputs and outputs can be assumed to be accessible, hence it suffices to carry out combinational test generation for each transition independently. Then, the algorithm in the last section can be employed to generate a functional test sequence.

For the input/output specification $\{(\langle I_i' \rangle / \langle O_i' \rangle) | i \in \mathcal{I}\}$ corresponding to a transition i, the sequence of steps of our test generation strategy can be described as in the following subsections.

4.1 Exact Simulation

The exact simulation can be performed by improving on the results of the inexact simulation using a line justification procedure that is commonly used in automatic test pattern generation [9]. For a node (line)

```
Logic_Simulate(C:circuit, B:input cube) {
  3_value_simulate(C,B);
  For all the gates in the cone of specified outputs {
    Create a list L of gate output nodes with X value
    sorted in order of their level from input to output }
  While L is non-empty{
      Remove node N at the head of the list L
      If ¬Justify(N, 0) then {
        Assign 1 to N;
        Carry out deterministic implications and update L;}
      Else If ¬Justify(N, 1) then {
        Assign 0 to N;
        Carry out deterministic implications and update L;}}
  For each primary output Z with specified value v {
      If ¬Justify(Z, v̄) then (report design error)}}
```

Fig. 3. An algorithm to do exact three-value logic simulation.

N in the circuit, the process Justify(N, v) determines if there is an input vector contained in the input cube that would set node N to the binary value v. For each node N with an X value after three-value simulation, if the call to Justify$(N, 0)$ fails we can immediately change the X value to 1 because it is not possible to justify a 0 value at node N by *any* setting of the unspecified inputs. Otherwise, we make the call Justify$(N, 1)$. If this fails, the node can be set to 0, otherwise, it must remain as X. Since the number of X values is bounded by the circuit size, at most a linear number of calls to Justify is necessary for the exact simulation.

This idea is incorporated in the algorithm shown in Figure 3. After the (inexact) three-value simulation, the algorithm collects all gate output nodes with X value that are in the cone of the specified outputs. These are tested for a constant value as above in order of their level from input to output. Whenever a node value changes, deterministic implications of the change are propagated to other nodes in the circuit and the list of remaining X nodes is pruned accordingly. In the final step, the algorithm checks for any discrepancies in the primary output values between the specification and exact simulation. In that case, a design error is detected independent of the settings of X values on the input.

Example: The circuit shown in Figure 4 will be used as a running example. For the input cube shown in the figure, assume both outputs are specified to be 1. Figure 4(a) shows the signal values after the (inexact) three-value simulation upon which the following sorted list L will be created:

$$L = \{k, l, m, q, r, s\}$$

It is possible to justify both 0 and 1 on k. Therefore this node retains its X value. The same is true of node l. However, Justify(m,0) fails therefore m is assigned

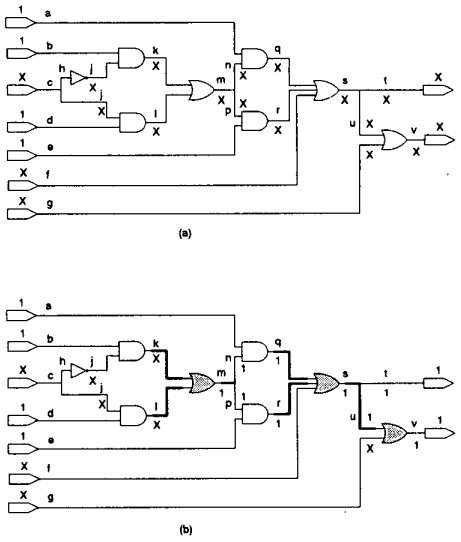

Fig. 4. Three-valued vs. exact simulation

constant 1 and lines n, p, q, r, s, t, u, and v are also assigned 1 by deterministic implication. As a result, the list L is pruned and becomes null, completing the while loop. The result is shown in Figure 4(b). The primary-output check in the last step succeeds as the PO values after exact simulation match the specification, hence no design errors are revealed at this stage.

4.2 Border Gate Identification

Border gates are identified via simulation. The example in Figure 4(b) has three border gates that are shown highlighted.

4.3 Fault List Generation

The fault list is generated following the Observation in Section 3 with some exceptions. We include the faults at the inputs of border gates of Types 0 and 1 as stated. However, for ease of computation, we include a superset of the fanout stems indicated in the Observation. Instead of verifying that a binary-valued fanout stem is included in the cone of all dominating inputs of a Type-2 border gate, we include all binary-valued stems in the cone of any border gate.

For the three border gates in the running example, the faults on the following lines will be included: k, l, q, r, and u. In addition, because the constant-valued stem m is in the input cone of q and r, the fault on line m will also be in the fault list.

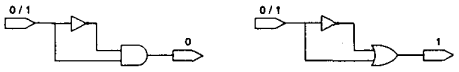

Fig. 5. Structural change to constrain input value.

4.4 Constrained Test Generation

The test generation must be carried out under input constraints; only the unspecified values in the input cube can be changed during test generation. It is possible to modify a PODEM-like algorithm that searches for a solution on a decision tree to allow branching and backtracking only on the unconstrained inputs. We accomplish the same goal by running a standard test generator [10] on a modified circuit that constrains the inputs internally (see Figure 5). A greedy approach is used to cover as many faults as possible by a single test vector before considering another vector in the input cube.

For the running example, the fault on line k is detected by the test cube $abcdefg = 110110X$ which also detects the fault on line m. Further expanding the test cube to 1101100 detects the fault on line u. Similarly, the test 1111100 detects faults on lines l m and u. The faults on lines q and r are not detectable by any vector in the original input cube. Therefore, only two vectors in the input cube cover all the faults detectable by all eight vectors included in the cube. There are 12 such faults on lines $b, d, h, i, j, k, l, m, s, t, u$, and v.

5. Experimental Results

We implemented the test generation described in the previous section and conducted experiments using a representative sample of 12 FSMs included in the 1991 logic synthesis benchmarks. We excluded from consideration small machines and those that include very few or no don't cares in their transitions because our approach does not provide any additional benefit in these cases.

The structural representations for the FSMs were produced using SIS [11] to simplify and synthesize the circuits using the *rugged script*. Technology mapping was limited to four-input simple gates. Each transition was expanded into one or more input vectors using the border gate approach and a shortest tour was obtained to cover all the resulting transitions. These tests are referred to as BG in reporting the results.

For comparison, we also obtained simpler functional tests (hereafter referred to as ST) in which successive randomly-generated tours (independent from the BG tours) were merged so that the tour length matched the tour length of the BG test set. The don't cares were randomly-filled in this case. Initially this test is equivalent to the functional test of Karam and Saucier [3] but expanded with additional tours to match the BG test length.

In the first set of experiments, we compared the BG and ST tests for their coverage of manufacturing faults. To this end, the tests were applied as sequences of vectors corresponding to their respective tours, and their coverage was evaluated using the HOPE fault simulator [13]. The results are presented in Table I. For each circuit the Table shows the number of states in the FSM followed by circuit statistics giving the number of primary inputs, primary outputs, gates, flip-flops, and number of faults. The last three columns give the test length and the comparison of the fault coverage for SAF faults. It will be seen that the coverage of the BG tests is consistently higher.

In the second set of experiments, the tests were evaluated for their coverage of design verification errors. As explained in Section 2, for design verification it is enough to apply the tests on the underlying combinational logic circuit. A recent program, ESIM [12], was used for this evaluation. This simulator can produce coverage of a test for single design errors of the following kinds: gate substitution errors (GSE), gate count errors (GCEs), input count errors (ICEs), and wrong input errors (WIEs). The GSE class is further subdivided into errors of single input gates (SIGSEs) and multiple input gates (MIGSEs). The GCE class is also divided into two subclasses corresponding to extra or missing gates (EGE and MGE, respectively). Similarly there are two subclasses, EIEs and MIEs corresponding to the class ICE.

Table II shows the results for the coverage of design errors. For each circuit, the test lengths are identical to the test lengths shown in Table I. This is followed by the coverage of the various classes of design errors. The results show that the BG tests cover more design errors than the ST tests for a majority of the tested circuits.

6. Conclusion

The fault model and the border-gate approach to test generation allows a unified approach to test generation for detecting design errors and manufacturing faults. The manufacturing tests are functional hence can be applied at the rated speed of the circuit. The results on the benchmark circuit show that our tests provide a high coverage for the design errors and SAF faults.

Acknowledgments: This work was supported by the NSF Grant No. CCR-9971167 and the University of

TABLE I

FAULT COVERAGE

Circ	# Sta	# PI	# PO	Comb Gates	# FF	Total Flts	Test Len	% Flt Cov ST	% Flt Cov BG
cse	16	7	7	135	4	368	516	97.8	99.1
ex1	18	8	19	121	5	366	925	99.7	99.7
ex6	8	5	8	68	3	189	70	97.8	98.4
keyb	19	7	2	158	5	396	705	86.8	99.7
opus	9	5	6	66	4	191	88	98.9	100.0
s386	13	7	7	78	4	233	333	97.8	97.8
s510	47	19	7	247	6	661	380	99.0	100.0
s820	24	18	19	170	5	485	2896	97.3	99.1
s832	24	18	19	164	5	485	3075	97.3	99.3
sand	32	11	9	354	5	936	978	98.3	99.5
sse	13	7	7	83	4	231	369	99.5	100.0
tma	18	4	6	137	5	364	299	99.4	100.0

TABLE II

COVERAGE OF DESIGN ERRORS.

Circuit	% Detected GSEs				% Detected GCEs				% Detected ICEs				% Detected WIEs	
	SIGSEs		MIGSEs		EGEs		MGEs		EIEs		MIEs			
	ST	BG	ST	BG	ST	BG	ST	BG	ST	BG	ST	BG	ST	BG
cse	97.1	97.1	91.3	94.7	100	98.5	87.9	91.6	76.2	89.2	54.3	63.4	82.6	91.8
ex1	100	100	94.9	97.1	100	100	91.2	94.3	90.9	99.0	69.1	97.1	96.4	98.6
ex6	96.8	95.1	94.1	94.5	100	100	85.1	90.2	75.7	81.7	64.0	61.9	80.0	85.1
keyb	90.3	96.8	90.5	93.4	97.8	100	82.2	89.2	77.3	89.3	55.7	63.7	81.7	90.1
opus	99.2	92.8	94.0	90.8	100	100	88.6	84.3	86.3	65.8	66.7	51.8	92.6	78.1
s386	99.3	96.7	94.2	94.9	100	100	92.4	91.6	88.1	87.4	65.5	65.0	93.6	91.3
s510	99.5	96.4	95.2	94.7	99.3	98.6	89.7	90.1	88.9	87.2	63.5	64.5	91.4	90.3
s820	98.7	99.6	95.3	96.8	100	100	92.2	96.3	93.3	96.6	68.6	76.6	95.7	98.1
s832	98.7	99.6	95.2	96.0	100	100	92.4	96.4	91.8	93.8	69.1	74.2	95.0	96.3
sand	99.7	98.5	95.5	95.3	99.5	99.5	92.2	92.8	95.0	94.5	67.7	67.2	96.0	95.4
sse	99.4	96.8	91.7	94.0	100	97.4	91.1	94.3	88.3	86.9	63.8	68.6	93.6	90.7
tma	98.3	99.1	94.3	96.3	100	98.7	87.5	89.6	85.7	90.3	61.0	70.2	88.7	92.6

Nebraska-Lincoln Center for Communication and Information Science. We are grateful to Dr. Hussain Al-Asaad for making ESIM available to us.

References

[1] K.-T. Cheng and J.-Y. Jou, "Functional test generation for finite state machines," in *Proceedings International Test Conference*, pp. 162–168, 1990.

[2] I. Pomeranz and S. M. Reddy, "Test generation for multiple state-table faults in finite state machines," *IEEE Transactions on Computers*, vol. 46, pp. 783–794, July 1997.

[3] M. Karam and G. Saucier, "Functional versus random test generation for sequential circuits," *Jour. Electronic Testing: Theory and Applications*, vol. 4, pp. 33–41, 1993.

[4] I. Pomeranz and S. M. Reddy, "On achieving complete fault coverage for sequential machines," *IEEE Transactions on Computer Aided Design*, pp. 378–386, March 1994.

[5] J. B. Adams and D. S. Hochbaum, "A new and fast approach to very large scale integrated sequential circuit test generation," *Operations Research*, vol. 45, pp. 842–856, November-December 1997.

[6] M. Weiss, S. C. Seth, S. Mehta, and K. L. Einspahr, "Exploiting don't cares to enhance functional tests," in *Proceedings International Test Conference*, 2000. To appear.

[7] P. Goel, "RAPS test pattern generator," *IBM Technical Disclosure Bulletin*, vol. 21, pp. 2787–2791, December 1978.

[8] M. Abramovici, J. J. Kulikowski, P. R. Menon, and D. T. Miller, "SMART and FAST: Test generation for VLSI scan-design circuits," *IEEE Design and Test*, pp. 43–54, August 1986.

[9] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Transactions on Computers*, vol. C-30, pp. 215–222, March 1981.

[10] H. K. Lee and D. S. Ha, "On the generation of test patterns for combinational circuits," Technical Report 12-93, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, College Station, TX 77840 USA, 1993.

[11] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Memorandum UCB/ERL M92/41, University of California, Berkeley, University of California, Berkeley, CA 94720 USA, May 1992.

[12] H. Al-Asaad and J. P. Hayes, "ESIM: A multimodel design error and fault simulator for logic circuits," in *Proceedings of the VLSI Test Symposium*, pp. 221–228, 2000.

[13] H. Lee and D. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," in *Proceedings 29th Design Automation Conference*, pp. 336–340, June 1992.

[14] J. Dworak, M. R. Grimaila, S. Lee, L.-C. Wang, and M. Mercer, "Modeling the probability of defect excitation for a commercial IC with implications for stuck-at fault-based ATPG strategies," in *Proceedings International Test Conference*, pp. 1031–1037, 1998.