Finite State Machine Testing Based on Growth and Disappearance Faults

M. K. Srinivas
CAD Laboratory
Indian Inst. of Science
Bangalore 560012, India

James Jacob
Department of ECE
Indian Inst. of Science
Bangalore 560012, India

Vishwani D. Agrawal
AT&T Bell Laboratories
Murray Hill, NJ 07974
USA

Abstract

We present a novel approach to generate functional test sequences for synchronous sequential non-scan circuits. The method is applicable when the functional description of the circuit can be obtained in the cubical form or a Personality Matrix (PM). The faults are modeled as growth and disappearance faults in the cubical description of the irredundant combinational function of the finite state machine (FSM). Considering the combinational logic alone, test vectors for these faults are efficiently derived using a cube-based method developed for programmable logic arrays (PLAs). It is shown that these tests cover 100% of stuck type faults in any irredundant two-level implementation and in the multi-level implementations obtained through testability preserving transformations. To derive tests for the sequential circuit, we represent it as an iterative array of the combinational logic whose PM is modified according to the fault. We give new PM based algorithms to obtain state justification and fault propagation sequences. Thus, the cube based PLA algorithm is extended to obtain the entire test sequence. Experimental results on MCNC synthesis benchmark FSMs and some ISCAS89 sequential circuits show that our approach can efficiently obtain functional test sequences which give very high coverage of stuck faults in specific implementations. The method has the added attraction that the functional test sequences are implementation independent and they can be obtained even when details of specific implementation are unavailable or unknown.

1 Introduction

The *growth* (G) and *disappearance* (D) faults in the combinational function of a circuit are a subset of the faults normally modeled in the programmable logic array (PLA) implementation [19]. It is known that the tests for G and D faults cover all stuck faults in any two level implementation of the combinational logic [12]. For certain synthesis styles [11, 17], these tests will also cover *all* single stuck faults in the multi-level combinational circuit.

The main contribution of this paper is a sequential circuit test generation algorithm based on the G and D fault model and its implementation. Many sequential circuit test generators use the time-frame expansion method where the circuit is represented as an iterative array of its combinational logic [1]. At the core of such a method there usually is a combinational test generation algorithm. In order to find a test sequence, the test generator repeatedly uses the combinational algorithm. Thus, the overall efficiency depends upon how well this algorithm performs. We model the combinational logic at the functional level by its personality matrix and develop an efficient cube based test generation algorithm [14] to obtain test sequences for G and D faults in the finite state machine (FSM).

Some recent approaches [5, 16] to functional test generation for sequential circuits rely on the transition fault model. Drawbacks of these approaches are that the number of single transition faults can be very large even for relatively small machines and it may often be necessary to consider multiple transition faults to achieve adequate single stuck fault coverage. In our approach, the number of functional (G and D) faults is quite reasonable and is of the same order as the number of single stuck faults in gate level implementations of the sequential function. Further, just like other functional approaches, our method also generates implementation independent test sequences. These tests have been shown to achieve high fault coverage of stuck faults in specific multi-level implementations and the test generation can be performed much faster compared to the conventional gate level methods.

Another approach, reported by Ghosh *et al* [10], also uses a cube based technique for justification and propagation on the fault free FSM, employing both the ON and OFF sets of the primary outputs and next state outputs. In our approach we use the faulty FSM for state justification and fault propagation to generate valid test sequences. We only require the ON sets of the primary output and next state functions. Our approach of functional test generation targets G and D faults in the extracted PLA and the test sequences can be applied to any multi-level implementation of the sequential function. The method of Ghosh *et al* [10] generates test sequences targeting stuck faults in a specific implementation. In their approach justification sequences start from a reset state, whereas we start from the last state of the previous test sequence.

Our functional test generation method is general but is particularly suited for the automatic synthesis environment. In the synthesis of FSMs, after the state

0731-3071/92 $03.00 © 1992 IEEE

assignment is done, the circuit is described as a combinational function. The description at this point is often in the form of Boolean *cubes* and resembles the functional specification of a PLA in the ***personality matrix*** form. A two-level non-redundant form, that is easily obtained using the available tools [2], is the input to our test generator. For the tests to retain their fault coverage, it is preferable to use only the testability preserving transformations [11, 17] in the synthesis of multi-level logic from the two-level minimized form. In cases where a two-level functional description is not available, we can use a cover enumeration technique [15] to obtain the personality matrix from the gate-level description of the sequential circuit. Such a strategy has been employed to extract the two-level descriptions of some ISCAS89 sequential benchmarks for which results are reported in this paper.

There are cases like the arithmetic or parity functions where the number of cubes in the two level sum-of-products form is exponential in the number of primary inputs. Our method, presently cannot handle these cases efficiently. However, the technique can be extended to large gate level combinational and sequential circuits if we partition them into interconnection of moderately sized functional blocks. This approach is under investigation.

2 Background

A broad outline of our algorithm for FSM test generation without the implementation results appeared in a recent paper [12]. For completeness, in this section, we repeat certain aspects of the PLA test generation methodology. Figure 1 (a) shows the personality matrix (PM) description of the combinational portion of an example FSM having two primary inputs (PIs), R and I, one flip-flop (FF) and one primary output (PO), C. The FF output, that feeds back into the combinational logic, is the present state input, PS, and the FF input is the next state output, NS, of the combinational logic. Figure 1 (b) shows the Karnaugh maps of the PO and next state functions. Figure 2 shows a two-level AND-OR implementation of the FSM.

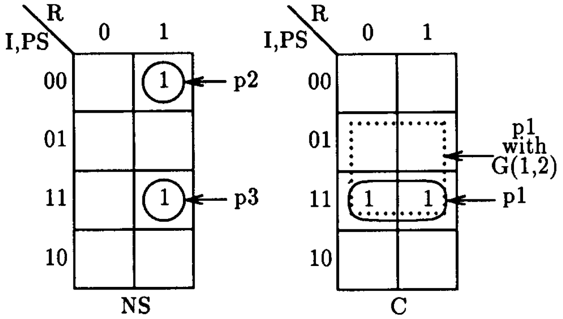
The personality matrix (PM) consists of two arrays: the AND array and the OR array. The cubes or product terms in the AND array are denoted as p_1, p_2 and p_3. A cube is a conjunction of literals representing PI and present state signals. In this case, $p_1 = I.PS$, $p_2 = R.\bar{I}.\overline{PS}$ and $p_3 = R.I.PS$. The output functions realized are given by:

$$NS = p_2 + p_3 = R.\bar{I}.\overline{PS} + R.I.PS;$$
$$C = p_1 = I.PS$$

A missing literal in a product term causes a ***growth*** (G) fault. If the literal corresponding to the j^{th} input x_j is missing from product term p_i, the corresponding G fault is denoted as G(i, j). For example, if the literal I is missing from product term p_1, then this product term will grow as shown by the dotted lines in the Karnaugh map in Figure 1 (b). This is the fault G(1, 2). A missing product term from an output function in the OR array causes a ***disappearance*** (D) fault. If

	AND array			OR array	
Product Terms	Inputs			Outputs	
	R	I	PS	NS	C
p1	x	1	1	0	1
p2	1	0	0	1	0
p3	1	1	1	1	0

(a) Personality matrix (PM)



(b) Karnaugh maps of combinational functions

Figure 1: Functional description of an example FSM

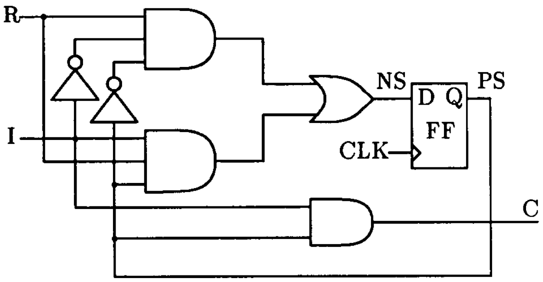


Figure 2: A two-level circuit for the FSM of Figure 1

the product term p_i is missing from the k^{th} output z_k, the corresponding D fault is denoted as D(i, k). For example, the fault D(2, 1) will cause the product term p_2 to vanish from the Karnaugh map of the first output function NS in Figure 1 (b).

For the G fault G(i, j), we define the ***candidate test cube*** (CTC) as the cube p_i with the j^{th} input complemented. For the D fault D(i, k) the CTC is defined as the cube p_i itself. For a personality matrix PM we define PTLIST(i, k) as the set of product terms connected to function z_k excluding p_i. For the PM in Figure 1, PTLIST(2, 1) = p_3 = {111}.

We assume that the reader is familiar with the basic cube operations [9] such as union ($\sqcup$), intersection ($\sqcap$) and set difference or sharp (#). Now the tests for the fault G(i, j) and D(i, k) detected on output z_k (provided p_i is connected to z_k) can be given as CTC # PTLIST(i, k), where the appropriate definition of CTC is used. A G fault may be detectable on any output fed by the affected product term but a D fault can only be detected at the output whose function is affected by the fault.

Consider the fault G(2, 3) in the PM of Figure 1. CTC = 101; PTLIST(2, 1) = p_3 = 111. Hence test for G(2, 3) = CTC # PTLIST(2, 1) = {101}. The computation of tests for G and D faults using the above method is straightforward and allows very efficient implementation [19].

3 Fault Model

The functional faults described above are a subset of the crosspoint faults, commonly used in PLA test generation [19]. Of the four types of crosspoint faults, namely, growth (G), shrinkage (S), appearance (A) and disappearance (D) faults, we have chosen the G and D faults that constitute the missing crosspoint faults, as the target faults for FSMs.

The usefulness of the G and D faults stems from their ability to model stuck faults in irredundant two-level circuits. It has been shown that all single stuck faults in an irredundant two-level single/multiple output circuit will be detected by the tests for G and D faults in the equivalent PLA provided the tests set each primary output to 0 at least once [13]. A natural question then is whether the G-D tests, that is the tests for all G and D faults, will also detect all detectable multiple stuck faults. It is known that redundant multiple stuck faults can exist even in single stuck fault irredundant two-level multi-output circuits. An example of such a circuit is given by Bryan *et al* [4]. It is also known that in an irredundant two-level single output circuit, a test set for all single stuck faults will detect all multiple stuck faults [18]. In a recent paper [13], it has been shown by a simple counter-example that this result is not *directly* applicable to irredundant multiple-output circuits.

The detection of multiple G and D faults (which model multiple stuck faults in the two-level circuit) by a test set for single G and D faults is important in the present case since some stuck faults in the multi-level implementation may map onto multiple G and D faults in the equivalent PLA. It is known [20] that in a G-D irredundant PLA, a test set for all single G and D faults will detect all multiple G and D faults, provided we can find an output ordering $z_1..z_q$ among the q outputs such that all G faults in the cone of output z_j are detected via one or more outputs $z_1..z_i$ ($1 \leq i \leq j \leq q$). This means that all faults in the cone of output z_1 will be detected on z_1. All faults in the cone of z_2 will be detected via z_2 except the ones already detected via z_1, and so forth. A similar result holds for stuck faults in multi-output circuits [13].

Many logic minimization algorithms used in synthesis yield single stuck fault irredundant two-level representations in the personality matrix form. Hence a test set for all single G and D faults, derived such that the output ordering discussed above is satisfied, will indeed detect all multiple stuck faults in any irredundant two-level gate implementation of the combinational function. There is also a high probability that this test set will detect most single stuck faults in a multi-level implementation as these faults are likely to have an equivalent multi-fault representation in the two-level circuit. If, however, the multi-level circuit is obtained by testability preserving transformations on the single-output minimized two-level form [11, 17], the tests for G-D faults will continue to cover all single and multiple stuck faults in the synthesized combinational circuit.

4 Test Generation for Combinational Circuits

We have implemented a new test generation and fault simulation program, GDCOMB, in C language. Tests are generated using the algorithm of Jacob and Biswas [14]. Fault simulation, performed after each test vector is generated, involves finding the Hamming distance between the test vector and the product terms. A G fault (D fault) on a product term is detected if the distance between the product term cube and the test vector is one (zero) and no other cube with a distance zero from that product term is connected to the same output.

We employed GDCOMB to derive tests from the personality matrix description of the combinational portion of six synthesis benchmark circuits. These results are given in Table 1. GDCOMB derived vectors to cover all G and D faults. These vectors were then used to simulate all *collapsed* single stuck faults in multi-level implementations of circuits. The coverage, as shown in Table 1, was 100% for all circuits.

Table 1 also gives the results of test generation for stuck faults in multi-level circuits by a gate level test generator, Gentest [6]. While both test generators could cover all faults, the run times of GDCOMB are significantly better. Vector sets of Gentest are, however, smaller. This is because the vector sets of GDCOMB are independent of the implementation. Such implementation-independent tests can also be derived from Gentest if vectors are generated for all single stuck faults in two-level AND-OR circuits. The fault set size and vector set size then will be comparable to those of the G-D faults and GDCOMB tests but the run time of Gentest will be even higher than that given in Table 1. The use of test vectors generated

Table 1: Test Generation for Combinational Part of FSMs (SUN Sparc 1)

Circuit Name	PI, PO, Prod. Terms	Personality Matrix			Multi-level Implementation				
		GDCOMB					Gentest		
		G-D Faults	No. of Vect.	CPU Sec.	Stuck Faults		No. of Vect.	Cov %	CPU Sec.
					Total	Cov %			
bbara	7, 5, 26	126	44	0.01	132	100	23	100	0.8
dk14	6, 8, 51	239	44	0.01	244	100	30	100	1.8
dk16	7, 8, 104	625	106	0.10	526	100	73	100	8.6
planet	13, 25, 235	1341	204	0.50	1066	100	108	100	25.9
sand	16, 14, 228	1547	346	1.00	1087	100	139	100	30.0
styr	14, 15, 228	1678	369	1.00	1127	100	149	100	34.4

from two-level AND-OR description was suggested by Dave and Patel [8].

The multi-level combinational circuits were synthesized from the ***single-output minimized*** two-level description, employing ***algebraic*** factorization and a simple technology mapping scheme that uses only AND and OR gates of up to four inputs and inverters. The synthesis system MIS [3] was used in our experiments. Since only testability preserving transformations [11, 17] were employed, the 100% fault coverage was expected. The two-level and multi-level combinational circuits of the FSMs were irredundant with respect to single stuck faults.

To examine the importance of testability preserving transformations in synthesis, we experimented with the circuit ***styr***. A multi-level implementation of this circuit was synthesized from a multiple-output minimized personality matrix description of the function, and a technology mapping that used a standard cell library. The multiple output minimized PM of ***styr*** had only 118 cubes as compared to 228 in the single-output minimized version. GDCOMB obtained a test set of 391 vectors for the 1239 G-D faults in the PM description in 0.7 seconds. However the vectors covered only 99.45% (i.e., 1085 of 1091) of detectable stuck-faults in this multi-level implementation.

5 Test generation for FSMs

We use an extension of the PLA test generation method described in Section 2 to derive tests for the G and D faults in the FSM. The algorithm consists of the following three steps:

Step 1: Combinational Test generation.
Step 2: State Justification.
Step 3: Fault Propagation.

We sandwich the combinational test vector from Step 1 (derived using the cube based algorithm described in Section 2) between the state justification and fault propagation sequences to obtain a complete test sequence for the G or D fault under consideration. A functional fault simulator for G and D faults was implemented and was used to reduce the fault list after the test sequence for a fault is generated.

5.1 State Justification

The combinational test generation procedure, as implemented in GDCOMB, produces a set of test cubes C_t with possible ***don't care*** entries for some present state (PS) bits. Any one of these vectors is sufficient to activate the fault in the present time frame t. The state S_t corresponding to any chosen vector must be justified to the state S_p at which the FSM was left after the application of the last vector in the previous sequence. The state for the first fault considered is justified to the reset state, assuming that the machine can be reset at power up even under faulty conditions. This assumption, used for simplicity in the present implementation, is not a basic limitation of our technique and can be removed in the future implementations.

Justification involves finding a sequence of vectors that will bring the FSM from the state S_p to the state S_t. In our implementation, reverse time processing is used to generate a justification sequence, starting from current state S_t, back to S_p. If any of the vectors generated in time frame t has the present state S_t which covers the state S_p, we do not have to generate a justification sequence for the fault under consideration. If no such cube exists, first we check if a single vector justification sequence exists. This can be done by finding all input cubes that have the fanin states of S_t (present state part) in the previous time frame $t-1$. These cubes can be found by simple cube intersection ($\sqcap$) and sharp (#) operations on the ON set cubes of the next state functions corresponding to the state S_t.

As an example, suppose we want to find all input cubes that will set the three next state signals in $t-1$ as $NS_1 = 1$, $NS_2 = 0$ and $NS_3 = 1$, i.e., $S_t = 101$. The input cubes that have the fanin states of S_t are given by $NS_1 \sqcap \overline{NS_2} \sqcap NS_3 = (NS_1 \# NS_2) \sqcap NS_3$, where NS_1, NS_2 and NS_3 are the ON sets of the corresponding next state lines. Once we find the fanin cubes C_{t-1} of S_t, we search C_{t-1} for a cube whose present state portion covers S_p. If such a cube exists in C_{t-1}, we have found a single vector justification sequence. Otherwise, we heuristically select a cube in C_{t-1} whose present state part S_{t-1} does not subsume S_t (to avoid self loops), find the fanin cubes C_{t-2} of S_{t-1} and search for a cube in C_{t-2} whose present state portion covers S_p. This process is continued until either we reach a predefined limit on the length of the justification sequence or a user defined time limit or the set of fanin cubes for a given S_{t-x} becomes empty, at which point backtracking is started. Backtracking is done by advancing the time frame forward and choosing the next available cube among the fanin cubes to be justified. If no more cubes are available, advancing the time frames continues until we exhaust all cubes in C_t of the time frame t.

One significant advantage of our G and D fault model is that the effect of a fault can be represented by a single bit change in one of the cubes constituting the ON set of the affected function. For example, the effect of a G fault G(i, j) is to change the j^{th} variable in cube p_i to x from 0 or 1. Similarly, the effect of a D fault D(i, k) is to change the k^{th} entry in the output part of cube p_i from 1 to 0. A G fault causes the expansion of the affected cube in the functions fed by it, as seen on the Karnaugh map (see Figure 1 (b)). A D fault will cause a cube to disappear from the affected function and thus cause a contraction of the function as seen on Karnaugh map.

5.2 Fault Propagation

This step is required if the initial combinational test vector generated for the target fault does not propagate the effect of the fault to any PO but only to one or more next state lines. We will illustrate the algorithm using the example FSM of Figure 1. Consider the G fault G(2, 3) in the FSM. Step 1 yields the combinational test vector 101 as we derived in Section 2. This implies that 10 is the test vector and the machine should be in state 1 for this test to be effective. In step 3 we require a sequence that will propagate the fault effect to a PO. Since the fault effect has reached the next state output NS we look for a product term of a PO function which uses the PS input corresponding to the output NS and derive the input conditions to observe this product term signal at the PO. This can be achieved by simple cubical operations. Let p_i be the chosen product term which uses PS, connected to PO z_k and does not have conflict in other present state positions with the output on the corresponding next state lines. We derive PTLIST(i, k) for p_i with respect to output z_k. In order to avoid simultaneous propagation of D and $\bar{D}$ values to the PO, we change the bit corresponding to input PS in the cubes in PTLIST(i, k) to x if it differs from the value of the bit position for PS in p_i. PTLIST also consists of cubes that do not have any conflict with p_i and are connected to z_k. Now, $p_i\#$ PTLIST(i, k) will give the required sensitizing condition. In our example, p_1 is the only product term for the output C which is the only PO. PTLIST(1,2) is empty. Hence the propagation sequence is x1.

If during step 3, we cannot find a product term of a PO that uses the required present state variable, we propagate the fault effect to some other next state output before we finally reach a PO, taking care not to allow propagation to the same next state line as in a previous time frame. Since we implicitly search the function space, we may have to backtrack at certain steps, but the algorithm is complete and will eventually find a test sequence, if one exists. To check for fault masking due to multiple path propagation, a fault simulation just for the fault under consideration is done for every propagation vector as soon as it is generated. If there is fault masking, then a different product term is chosen for propagation.

For fault propagation, we always use the ON set of the faulty function, if the function is affected by the fault. Note that we can obtain the ON set of a faulty function by a simple bit change in one of its cubes.

5.3 Fault Simulation

A simple fault simulation based on the single fault propagation technique is implemented. When a test sequence is found for a fault, the vectors in the sequence are run through the fault simulator. For each vector first a fault free logic simulation is carried out with the PI portion concatenated with the fault free state bits for the present portion portion. The next state bits (fault free state) is saved for consideration with the next vector. Next, for each fault, modifications are introduced in the PM and the simulation is repeated with the PI portion of the vector concatenated with the faulty state bits of the fault under consideration. The output responses are compared in the PO portion to check whether the fault is detected. If the fault is not detected the next state bits (faulty state) are saved for the particular fault, for consideration with the next vector.

Since we use the cubical description of the logic function, simulation requires finding the Hamming distance of the vector to product terms. The outputs of all those product terms that have a distance of zero with the vector will be set to logic 1, and all other outputs will be set to logic 0.

6 Experimental Results

We developed a C program, GDSEQ, to generate test sequences for PLA based FSMs and general sequential circuits whose combinational function can be obtained in personality matrix form. We experimented on six synthesis benchmark FSMs and 16 of the ISCAS89 benchmarks. The characteristics of these circuits are shown in Table 2. The first six circuits are the synthesis benchmarks and the personality matrices of these circuits were available. As two-level functional description for the ISCAS89 benchmarks is not available we extracted the ON set cubes of all PO and next state functions of these circuits. The SUN Sparc 2 run times for cube extraction using a Podem based technique [15] and single output minimization using Espresso [2] are given in the last two columns of Table 2.

The results obtained from GDSEQ are given in Table 3. In the synthesis benchmark circuits (*bbara* through *styr*), 100% of G and D faults were covered almost always. As stated earlier, a power-up reset was assumed only at the beginning of the test sequence. The coverage of G and D faults in ISCAS89 circuits was lower due to the time frame limit and the backtrack time limit used in the justification stage of the program.

Next, the GDSEQ vectors were used to simulate all collapsed single stuck faults in the multi-level gate

Table 2: Characteristics of Benchmark FSMs

FSM	No. of Inputs	No. of Outputs	No. of Flip-Flops	Extracted PLA		Multilevel Circuit		SUN Sparc 2 CPU s	
				Prod. Terms	G-D Flts.	No. of Gates	Stuck Flts.	Cube Enumeration	Minimization
bbara	4	2	3	26	126	42	132	-	-
dk14	3	5	3	51	239	85	244	-	-
dk16	2	3	5	104	625	182	526	-	-
planet	7	19	6	235	1341	360	1066	-	-
sand	11	9	5	228	1547	354	1087	-	-
styr	9	10	5	228	1678	379	1127	-	-
s27	4	1	3	15	47	10	32	0.12	0.14
s208	11	2	8	30	202	96	215	0.29	0.49
s298	3	6	14	68	309	119	308	0.96	0.96
s344	9	11	15	249	1495	160	342	2.39	2.37
s349	9	11	15	249	1495	161	350	2.42	2.37
s382	3	6	21	167	1080	158	399	2.32	1.62
s386	7	7	6	51	407	159	384	1.10	0.61
s400	3	6	21	167	1080	162	424	2.63	1.71
s444	3	6	21	167	1080	181	474	2.89	1.58
s510	19	7	6	109	580	211	564	1.58	0.75
s526	3	6	21	142	740	193	555	1.93	1.40
s526n	3	6	21	142	740	194	553	1.84	1.40
s820	18	19	5	126	969	289	850	0.92	2.98
s832	18	19	5	126	969	287	870	0.92	2.98
s1488	8	19	6	277	1876	653	1486	1.05	3.46
s1494	8	19	6	277	1876	647	1506	1.09	3.28

Table 3: Test Generation on Multilevel FSMs using Functional Vectors (SUN Sparc 2)

FSM	Personality Matrix			Multi-level Implementation						
	GDSEQ						Random	Gentest		
	No. of Vect.	G-D Flt. Cov %	TGen. CPU Sec.	FSim. CPU Sec.	Useful Vect.	Stuck Fault Cov %	Vector Cov %	No. of Vect.	Stuck Fault Cov %	CPU Sec.
bbara	151	100.0	0.17	0.37	150	100.0	78.0	120	100.0	10.0
dk14	89	100.0	0.12	0.39	79	100.0	93.8	64	100.0	10.0
dk16	191	100.0	1.56	1.44	179	100.0	97.9	346	100.0	1403.0
planet	644	99.9	50.80	5.39	624	100.0	99.0	509	100.0	8446.0
sand	1210	100.0	719.70	14.03	1056	100.0	96.7	580	100.0	22425.0
styr	1191	100.0	102.20	10.96	1191	100.0	74.5	754	100.0	35868.0
s27*	23	89.4	0.02	0.03	14	100.0	65.6	18	100.0	0.1
s208*	260	44.6	0.56	2.19	232	69.3	40.0	10	8.3	682.9
s298*	187	72.8	2.84	0.83	182	86.0	51.6	220	86.0	337.9
s344	129	26.1	700.50	1.16	129	90.6	87.4	80	96.2	79.9
s349	122	26.9	690.90	0.61	122	91.7	86.5	105	95.7	123.7
s382	754	39.8	580.48	4.42	753	87.2	14.5	3796	91.2	3165.2
s386*	317	64.1	1.66	1.75	296	81.7	50.0	201	81.7	1707.6
s400	572	35.2	343.77	4.21	543	81.6	14.3	1517	90.1	2654.7
s444	387	30.6	81.14	4.38	387	75.5	12.2	2441	89.2	5447.7
s510*	550	99.1	35.74	2.14	550	100.0	99.8	0	0.0	0.4
s526	330	44.9	306.44	4.95	330	67.6	9.9	2223	79.3	16787.7
s526n	312	41.5	312.50	6.18	257	65.6	9.9	3445	80.8	19574.4
s820*	1345	88.6	145.80	14.47	1345	94.0	37.3	392	76.0	16735.5
s832*	1527	88.4	96.05	17.98	1500	92.1	36.2	367	76.0	19006.0
s1488*	1502	83.5	116.49	43.85	1502	94.2	57.2	393	91.2	30114.6
s1494*	1529	85.1	115.04	43.85	1529	94.5	56.3	421	90.0	22877.0

implementations of FSMs. A differential fault simulator [7] was used for fault simulation. As shown in Table 3, these vectors gave 100% fault coverage for all six synthesis benchmarks. The combinational portion of these FSMs were obtained by the synthesis program MIS [3] as described in Section 4 and these FSMs were single stuck fault irredundant. It should be noted that though testability preserving transformations were employed in the synthesis of these benchmark FSMs, the theoretical result on the multifault coverage of such circuits is applicable only to combinational circuits [11, 17]. We do not have any theoretical basis to claim 100% stuck fault coverage for sequential circuits synthesized by testability preserving transformations.

The ISCAS89 benchmarks were not synthesized by us but were the same as obtained from MCNC and the GDSEQ vector fault coverages of stuck faults in these circuits are lower than the maximum obtainable. It should, however, be noticed that the stuck fault coverage is always higher than that of the G and D fault coverage. The *useful vectors* given in Table 3 were obtained when the vector set was truncated after the detection of the last fault. For comparison, the coverage of a set of random vectors having the same number of vectors as the useful vectors is also given in Table 3. The random vector coverage is consistently lower.

We used the sequential test pattern generator Gentest [6] to verify the efficiency of GDSEQ. Gentest is a gate-level test generator and uses the time frame expansion method. It has a differential fault simulator [7] to remove detected faults from the fault list after a test sequence is generated for a target fault. We generated test sequences for stuck faults in the multi-level implementations of FSMs. The vector set size, fault coverage, and CPU times for Gentest on SUN Sparc 2 are given in Table 3. Here no power up reset is assumed. Also, the actual fault coverage can be higher in some cases due to the *undetectable* faults identified by Gentest. It is clear that the total time required to generate the functional (G-D) vectors using GDSEQ and fault simulate them on multi-level FSMs is far less than the time taken by Gentest.

For nine of the sixteen ISCAS89 circuits (shown with an asterisk in Table 3) the GDSEQ vectors gave equal or higher fault coverage than that obtained by Gentest. This is due to the fact that some of the faults aborted by Gentest were detected by GDSEQ. Seven circuits have lower fault coverage for GDSEQ than that obtained by Gentest. For these circuits GDSEQ can be run with different values of time limit and frame limit per fault to generate more functional vectors to improve the multilevel stuck fault coverage. The lower fault coverages for these circuits may also be due to the fact that these circuits were not synthesized using testability preserving transformations. It may be noted that for two circuits, namely, s208 and s510, for which Gentest coverage was 8.3 and 0%, the GDSEQ vectors gave 69.3 and 100% coverage, respectively. This is due to the initialization difficulty in these circuits and the power up reset assumption of GDSEQ.

The CPU times for the functional approach and that of Gentest cannot be compared meaningfully for all ISCAS89 circuits since the coverages are different. However considering the nine circuits (shown by asterisk in Table 3) where the GDSEQ vectors gave equal or greater coverage than Gentest, the functional approach (test generation + simulation) is 1 to 500 times faster compared to Gentest.

To examine the advantage of using testability preserving transformations in the synthesis of sequential circuits, we experimented with the circuit *styr*. The multi-level combinational portion of *styr* synthesized from a multiple-output minimized personality matrix (as already described in Section 4) was used to construct the multi-level FSM. GDSEQ generated a test sequence of 1354 vectors for the 1239 G-D faults in multiple-output minimized PLA based FSM. These vectors gave only 98.9% coverage (i.e., 1080 of 1091) of detectable stuck faults in the multi-level implementation, as opposed to the 100% coverage obtained for the testable implementation reported in Table 3. Thus, the loss of fault coverage due to improper synthesis, though not large, is noticeable.

7 Conclusion

The model of growth and disappearance faults in the logic function of a finite state machine allows efficient test generation. We found that the functional test sequence derived by a prototype implementation of our test generation algorithm could achieve a very high coverage of single stuck faults in actual multi-level FSM implementations. We must, however emphasize the usefulness of testability preserving transformations in synthesis, as is evidenced by our experiment with the circuit *styr*. The functional fault model also allows us to generate tests that are independent of the specific logic implementation. A major advantage of this approach is that functional test generation combined with fault simulation is considerably faster than gate level algorithms that target stuck faults in a specific implementation. For the relatively few stuck faults that may not be detected by the functional test sequence, it is possible to generate additional tests using any gate level sequential circuit test generator.

For generating tests for large circuits with only the gate-level implementation given, it is not necessary to extract the functional behavior (personality matrix) of the entire circuit. A better approach may be to partition the circuit and solve the test generation problem for an interconnection of functional blocks.

References

[1] V. D. Agrawal and S. C. Seth, *Test Generation for VLSI Chips*, IEEE Computer Society Press, Los Alamitos, CA, 1988.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI synthesis*, Kluwer Academic Publishers, Boston, 1984.

[3] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. R. Wang, "MIS : A Multiple

Level Logic Optimization System", *IEEE Trans. on CAD*, vol. 6, pp. 1062-1081, November 1987.

[4] D. Bryan, F. Brglez and R. Lisanke, "Redundancy Identification and Removal," *Proc. Int. Workshop on Logic Synthesis*, Research Triangle Park, North Carolina, 1989.

[5] K-T. Cheng and J. Y. Jou, "Functional Test Generation for Finite State Machines", *Proc. Intl. Test Conf.*, pp 162-168, 1990.

[6] W. T. Cheng and T. J. Chakraborty, "GENTEST: An Automatic Test-Generation System for Sequential Circuits," *Computer*, vol. 22, pp. 43-49, April 1989.

[7] W. T. Cheng and M. L. Yu, "Differential Fault Simulation for Sequential Circuits," *Journal of Electronic Testing: Theory and Applications*, vol. 1, pp. 7-13, February 1990.

[8] U. Dave and J. H. Patel, "A Functional-level Test Generation Methodology using Two-level Representations," *Proc. 26th Design Automation Conference*, pp. 722- 725, 1989.

[9] D. L. Dietmeyer, *Logic Design of Digital Systems*, Allyn and Bacon, Boston, 1971.

[10] A. Ghosh, S. Devadas, and A. R. Newton, "Test Generation and Verification for Highly Sequential Circuits," *IEEE Trans. on CAD*, vol. 10, pp. 652-667, May 1991.

[11] G. Hachtel, R. Jacoby, K. Keutzer and C. Morrison, "On Properties of Algebraic Transformations an the Multifault Testability of Multilevel Logic," *Proc. Int. Conf. on Computer Aided Design*, pp. 422-425, 1989.

[12] J. Jacob and V. D. Agrawal, "Functional Test Generation for Sequential Circuits", *Proc. 5th Intl. Conf. VLSI Design*, Bangalore, India, pp 17-24, January 1992.

[13] J. Jacob and V. D. Agrawal, "Multiple Fault Detection in Two-Level Multi-output Circuits", *Jour. Electronic Testing: Theory and Applications*, vol. 3, May 1992.

[14] J. Jacob and N. N. Biswas, "PLATES: An Efficient PLA Test Pattern Generator," *Proc. Third Int. Workshop on VLSI Design*, Bangalore, India, pp. 147-154, January 1990.

[15] H. K. T. Ma and S. Devadas, "Logic Verification Algorithms and their Parallel Implementation", *IEEE Trans. on CAD*, vol. 8, pp 181-188, February 1989.

[16] I. Pomeranz and S M Reddy, "On Achieving a Complete Fault Coverage for Sequential Machines using the Transition Fault Model", *Proc. Design Automation Conf.*, 1991, pp 341-346.

[17] J. Rajski and J. Vasudevamurthy, "Testability Preserving Transformations in Multi-level Logic Synthesis," *Proc. Int. Test Conference*, pp. 265-273, 1990.

[18] D. R. Schertz and G. Metze, "A New Representation for Faults in Combinational Digital Circuits," *IEEE Trans. on Computers*, vol. C-21, pp. 858-866, August 1972.

[19] J. E. Smith, "Detection of Faults in Programmable Logic Arrays," *IEEE Trans. on Computers.*, vol. C-28, pp. 845-853, November 1979.

[20] J.E. Smith, "Author's Reply," *IEEE Trans. on Computers*, vol. C-35, p. 931, October 1986.