Fault Equivalence and Diagnostic Test Generation Using ATPG

Andreas Veneris, Robert Chang
University of Toronto
Dept ECE
Toronto, ON M5S 3G4
{veneris, rchang} @eecg.toronto.edu

Magdy S. Abadir
Motorola
7700 W. Parmer
Austin, TX 78729
m.abadir@motorola.com

Mandana Amiri
University of British Columbia
Dept ECE
Vancouver, BC V6T 1Z4
mandana@ece.ubc.ca

Abstract

Fault equivalence is an essential concept in digital design with significance in fault diagnosis, diagnostic test generation, testability analysis and logic synthesis. In this paper, an efficient algorithm to check whether two faults are equivalent is presented. If they are not equivalent, the algorithm returns a test vector that distinguishes them. The proposed approach is complete since for every pair of faults it either proves equivalence or it returns a distinguishing vector. This is performed with a simple hardware construction and a sequence of simulation/ATPG-based steps. Experiments on benchmark circuits demonstrate the competitiveness of the proposed method.

1 Introduction

Computing the complete set of fault equivalence classes in a combinational circuit is a classic problem in digital circuit design. Two faults are *functionally equivalent* (or *indistinguishable*) if no input test vector can distinguish them at primary outputs. Functional fault equivalence is a relation that allows faults in a circuit to be collapsed into disjoint sets of *equivalent fault classes*. Fault equivalence is essential in digital VLSI because it has significance in fault diagnosis [7], diagnostic test generation [5], testability analysis [7] nd logic synthesis [3] [7] [8] [9].

Methods to compute fault equivalence are classified as *structural* and *functional* [7]. Structural methods operate on the circuit graph to identify fault equivalence. These methods are fast but they have pessimistic results since they operate on fan-out free circuit regions only. Functional fault equivalence methods are more expensive but they typically identify more classes [1] [2]. These methods use logic implications and/or dominator information to prove equivalence. Since identifying logic implications is NP-hard [8], these methods do not utilize the complete set of logic implications and they may not return the complete set of fault equivalence classes.

In this paper, we present an efficient method that proves the equivalence of a pair of stuck-at faults. The method also returns a distinguishing vector if the faults are proven not to be equivalent. To simplify the discussion, we use the term "fault" to indicate a single stuck-at fault hereafter. To prove *fault equivalence* or perform *Diagnostic Automated Test Pattern Generation (DATPG)*, the method performs a sequence of simulation- and ATPG-based steps. It should be noted, ATPG is not exclusive to the method and other test generation and redundancy checking techniques (BDDs, SAT solvers etc) can be utilized.

The proposed methodology has a number of characteristics that make it attractive and practical. Unlike methods that alter existing ATPG tools [5], it uses conventional ATPG [4] [8] and a novel hardware construction to either prove equivalence of the fault pair or return a distinguishing vector. Therefore, it automatically benefits from advances in ATPG and remains straightforward to implement. It is also *complete* in the sense that for every fault pair it guarantees to prove equivalence or return a distinguishing vector, provided sufficient backtracking level in the ATPG engine. To the best of our knowledge, this is the first published results on the exact number of stuck-at fault equivalence classes for ISCAS'85 circuits.

The paper is organized as follows. Section 2 presents the two steps of the proposed functional fault collapsing and DATPG algorithm in terms of single stuck-at faults. Section 3 presents experiments and Section 4 concludes this work.

2 Fault Equivalence and DATPG

In this Section we present the *two* steps of the fault equivalence/DATPG method. The *first step* computes an approximation of fault equivalence classes using structural fault collapsing and input test vector simulation. Faults in the same class may or may not be equivalent, but faults in different classes are *guaranteed* to be not equivalent. The *second step* uses conventional ATPG and a novel hardware construction on pairs of faults in the same class as computed in Step 1 to either formally prove the faults are equivalent or perform DATPG. Therefore, more equivalent fault classes may be identified in this step.

2.1 Parallel Vector Simulation

The implementation starts with structural fault collapsing [7] to prove faults that are structurally proximal as equivalent so that only one representative fault from each such set needs be considered in later steps of the algorithm. Let set F be the complete set of representative faults. The faults in F are examined for equivalence by *Parallel Vector Simulation (PVS)*.

PVS is a simulation-based procedure that classifies these faults into potentially equivalence classes $F_1, F_2, \ldots, F_n$ with respect to an input test vector set T. PVS identifies (maps) two faults f_A and f_B as *potentially equivalent* if and only if f_A and f_B give the exact same primary output responses for each test vector from the set T. If faults f_A and f_B are potentially equivalent, they are placed into the same class F_i. Faults in different classes are guaranteed not to

```
Parallel_Vector_Simulation(C, F, T)

(1) Simulate test vectors in T and create
    indexed bit-lists at every circuit line
(2) For every fault f s-a-v on line l do
(3)     fault_signature=0
(4)     set bit-list of l to value v
(5)     simulate at fan-out cone of l
(6)     update fault_signature
(7)     restore bit-lists at l fan-out cone
(8) Group faults with same signatures in
    same class F_i, i=1 ... n
```

Figure 1: Pseudocode for PVS (Step 1)

be equivalent since they already have different responses for some vector(s) in T. In fact, this vector(s) is also a distinguished vector for the faults in these classes.

Pseudocode for PVS is given in Fig. 1. The input to PVS is a circuit C, the collapsed set of faults F and a set of input test vectors T. In experiments, T is a relatively small set of 100-1000 test vectors with high stuck-at fault coverage [6]. The output of PVS is a set of fault classes $F_1, F_2, \ldots, F_n$ such that two faults f_a and f_b are in the same class F_i if and only if they have the exact same responses for all vectors in T.

At first, PVS simulates in parallel [7] all test vectors in T and stores the logic value of each line in an indexed bit-list [9] (Fig. 1, line (1)). Next, for every stuck-at v fault $f \in F$ on line l, value v is injected on l and simulated at the fan-out cone of l. The primary output bit-lists are treated as integer signatures for fault f and test set T (lines 2-6) before values are restored in line 7. This process is repeated for every fault in F and faults with the same signature are grouped together in line 8.

2.2 Fault Equivalence and DATPG

Given a pair of faults (f_A, f_B) from the same class F_i, Step 2 performs a hardware construction and employs conventional ATPG to *formally* prove their equivalence or return a test vector to distinguishe them (DATPG). It should be noted, formal fault equivalence or DATPG is performed in an *atomic step* for each pair of faults. We now outline the theory and implementation of this step in detail.

Assume, without loss of generality, faults f_A s-a-0 and f_B s-a-1 on lines l_A and l_B of the circuit. To examine their equivalence, the algorithm attaches two multiplexers, MUX_A and MUX_B, with common select line S. The 1-input of MUX_A is line l_A while the 0-input of the multiplexer is a constant logic value 0. Intuitively, a constant 0 indicates a s-a-0 fault on l_A. Similarly, the 0-input of MUX_B is the original line l_B and the 1-input is a constant 1.

This construction allows us to simulate the original circuit under presence of fault f_A when $S = 0$ and the original circuit under presence of f_B when $S = 1$. Therefore, if ATPG for select line S s-a-0 (ATPG for select line S s-a-1 produces similar results) exhausts the solution space to return no test vector and reports that the fault on S is redundant, then (f_A, f_B) is an equivalent fault pair [9]. In other words, the original circuit under the presense of each of the (f_A, f_B) faults behaves identically. Otherwise, the input test vector returned is a vector that distinguishes the two faults.

To illustrate this process, we can employ pair of values to indicate

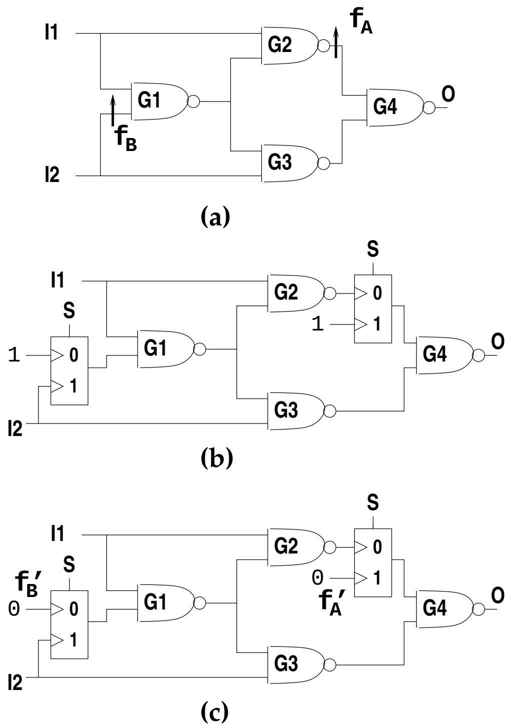

Figure 2: Circuits for Examples 1 and 2

simulation of *two* faulty circuits; one under the presence of f_A and the other under the presence of f_B. If the stuck-at fault on select line S is redundant, it implies that no $0/1$ and no $1/0$ value propagate(s) to any primary output. In other words, the two faulty circuits produce the same response for *all* input test vectors and the two faults are indistinguishable.

On the other hand, if a single $0/1$ ($1/0$) difference is propagated to a primary output, then one of the two faults is *guaranteed* to be detected. Which fault is detected depends on the logic simulation value; if logic simulation gives 0 then f_B (f_A) is detected at the primary output since circuit under presence of f_A (f_B) and logic simulation have identical values. Both faults are detected if appropriate $0/1$ or $1/0$ simulation values propagate at different primary outputs. The examples that follow illustrate the above procedures.

```
Fault_Equivalence_DATPG(C, F_1, ..., F_n)

( 1) flag=0
( 2) for i=1 to n do
( 3)     randomly select f from F_i
( 4)     for every f′ in Fi do
( 5)         perform the MUX construction
( 6)         if f′ not equivalent to f do
( 7)             flag=1
( 8)             place f′ in F_n+1
( 9)             store distinguishing vector
(10)      if flag=1
(11)         flag=0
(12)         n=n+1
```

Figure 3: Fault Equivalence and DATPG

Table 1: Parallel Vector Simulation (Step 1)

ckt name	# of initial faults	faults after collaps.	# ATOM vectors	# of fault classes after PVS: ATOM vectors	ATOM and 500 random	500 random	1000 random	CPU time (sec)
c432	798	419	110	371	417	413	418	0.11
c499	2434	1314	127	901	1076	1027	1092	0.59
c880	1770	940	133	857	889	853	868	0.17
c1355	2412	1302	192	1046	1088	1010	1079	0.89
c1908	1802	975	210	714	748	684	767	0.50
c2670	3177	1627	242	1141	1178	1160	1184	0.42
c3540	4116	2143	264	1408	1548	1541	1580	10.9
c5315	7042	3743	216	2971	3404	3381	3415	2.06
c6288	14303	7479	64	6397	6597	6593	6597	0.99
c7552	10081	5321	393	4186	4280	4180	4273	5.98

Example 1: Consider the circuit in Fig. 2(a) and assume two faults from the same class F_i $f_A = G_2 \rightarrow G_4$ and $f_B = I_2 \rightarrow G_1$ both s-a-1. To test their equivalence, we can place two multiplexers, shown as boxes in Fig. 2(b), with common select line S. The 0-input to the first multiplexer is line G_2 and the 1-input of that multiplexer is a logic 1 to represent the presence of a s-a-1 fault. Similarly, the 0-input of the second multiplexer is a logic 1 while I_2 feeds the other input. In both cases, the output of each multiplexer connects to the original output in the circuit. The reader can verify that when $S = 0$, we operate on a circuit equivalent to the one in Fig. 2(a) under the presence of f_A and when $S = 1$ we operate on a circuit equivalent to the one in Fig. 2(a) under the presence of f_B. ATPG for select line S s-a-0 returns the fault is redundant. This indicates that the two circuits are functionally equivalent which, in turn, confirms that (f_A, f_B) is an equivalent fault pair.

Example 2: Consider again circuit in Fig. 2(a) and faults $f'_A = G_2 \rightarrow G_4$ and $f'_B = I_2 \rightarrow G_1$ this time both stuck at logic 0. A similar multiplexer construction as in Fig. 2(b) gives circuit shown in Fig. 2(c). The difference is that a logic `0` is placed on appropriate multiplexer inputs to indicate a stuck-at-0 fault. ATPG on common select line S s-a-0 returns test vector $(I_1, I_2) = (0, 0)$. This proves that the fault on S is not redundant and faults f'_A and f'_B are not equivalent. This is true since the test vector returned detects fault f'_A but does not even excite fault f'_B. In this case, the construction returns a distinguishing vector for fault pair (f'_A, f'_B).

Fig. 3 contains pseudocode for Step 2. For each class F_i ($i = 1 \dots n$), a representative f is randomly selected. For each other member $f' \in F_i$, we perform the multiplexer construction to check whether f and f' are equivalent (lines 4-5). If they are not equivalent, f' (and all other such non-equivalent faults from F_i) is placed in new class F_{n+1} (lines 6-9) which will be examined later and the distinguishing vector is recorded. Observe, any such fault f' is guaranteed not to be equivalent with faults in any class $F_1, \dots, F_{i-1}, F_{i+1}, \dots, F_n$ by PVS. Faults placed in F_{n+1} may or may not be equivalent. Therefore, class F_{n+1} may get decomposed into new classes when it is examined later. The set of classes returned upon termination of the algorithm are also the exact fault equivalence classes for circuit C and fault set F.

3 Experiments

We implemented and ran the proposed method on an Ultra 5 SUN workstation with 128 Mb of memory. The details of the ATPG engine used can be found in [4][8]. We use a relatively low level 1 for recursive learning to provide a fair comparison with state-of-the-art diagnostic test generation tool DIATEST [5] and confirm the competitiveness of the approach. We evaluate the proposed techniques on ISCAS'85 combinational circuits optimized for area. Test vectors with high stuck-at fault coverage (ATOM vectors) are computed as in [6]. Run-times reported are in seconds.

Table 1 contains information about the first step of the algorithm. The first column of the table shows the circuit name and the second column contains the total number of stuck-at faults. This is roughly twice the number of lines, including branches. The third column shows the faults after structural fault collapsing [7].

We examine the performance of PVS with a set of random and stuck-at fault input test vectors. The number of ATOM vectors generated is shown in column 4. Columns 5–8 of Table 1 show the number of distinct fault classes upon termination of PVS for four different cases with respect to the test vector set T used: *(i)* ATOM vectors, *(ii)* ATOM vectors and 500 random vectors, *(iii)* 500 random vectors, and *(iv)* 1000 random vectors.

Intuitively, the more vectors we simulate the more accurate the results we expect in terms of number of classes, as discussed earlier. A study of the numbers indicates that a relatively small set of random vectors (case *(iv)*) gives sufficient resolution. In most cases, random simulation gives good resolution and there is little to gain with a pre-computed set of stuck-at fault test vectors.

This is also illustrated in Fig. 4 that depicts the number n of fault classes $F_1, F_2, \dots, F_n$ versus the number of random vectors simulated. It is seen that the number of fault classes converges with a relatively small number of vectors. We use the classes from case *(iv)* as input to Step 2. The last column of the table contains the total run-time for Step 1.

Table 2 contains information that pertain to Step 2. Given fault pair (f_A, f_B), it tests whether the two faults are equivalent and returns a distinguishing vector if they are not. Columns 2-4 of Table 2 show values that pertain to the case when the faults are equivalent. The total number of fault pairs checked and the number of final (complete) fault classes are found in columns 2 and 3.

The relative error for PVS (Step 1), a simulation-based process, when compared to the formal engine of Step 2 is found in column 4. It is seen that in many cases the relative error is rather small (less than 10%). This suggests that simulation of random vectors provides in most cases sufficient resolution to compute fault equivalence. Therefore, the designer is presented with a relatively small trade-off between time and accuracy.

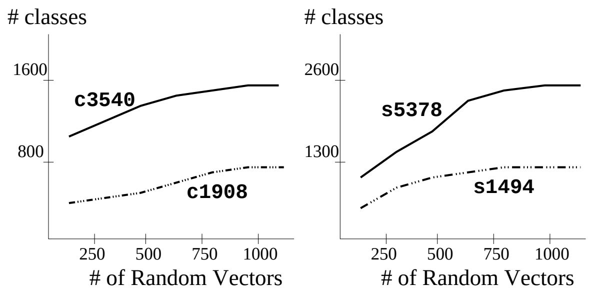

Figure 4: Performance of PVS

DATPG results are found in columns 5-7. Column 5 contains the number of distinguishable fault pairs checked. Columns 6 and 7 contain the *manner* these faults are detected. Recall from subsection 2.2, DATPG guarantees to detect one fault but it may detect both faults at *different* primary outputs. The numbers in these columns indicate that in as many as half of the cases, DATPG returns a test vector to distinguish *both* faults at different primary output (column 7). Column 8 of the Table 2 contains the average run-time for ATPG when the faults are equivalent (redundancy checking).

The last two columns in Table 2 provide comparison results with DIATEST [5] for the same list of fault pairs that are not equivalent. It is seen, that conventional ATPG and the hardware construction presented here provides an attractive alternative to a DATPG-specific engine. In fact, larger values of implication learning [8] will speed the ATPG tool and further improve performance of the method. This confirms the practicality of the approach that automatically benefits from advances in ATPG.

4 Conclusions

A method for fault equivalence and diagnostic test generation was presented. The method is practical since it uses conventional ATPG and a simple hardware construction to prove equivalence or return distinguishing test vectors. Experiments demonstrate its robustness, effectiveness and competitiveness.

Acknowledgments

The authors would like to thank Prof. W. K. Fuchs, Dr. E. Amyeen and S. Seyedi for their technical comments and support in this work.

References

[1] V. D. Agrawal, D. H. Baik, Y. C. Kim and K. K. Saluja, "Exclusive Test and its Applications to Fault Diagnosis," in *Proc. of IEEE Int'l Conf. on VLSI Design,* 2003.

[2] M. E. Amyeen, W. K. Fuchs, I. Pomeranz and V. Boppana, "Fault Equivalence identification using redundancy information and static and dynamic extraction," in *Proc. of IEEE VLSI Test Symposium, pp. 124-130,* 2001.

[3] S. C. Chang and M. Marek-Sadowska, "Perturb and Simplify: Multi-Level Boolean Network Optimizer," in *Proc. Int'l Conference on Computer-Aided Design, pp. 2-5,* 1994.

[4] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," in *IEEE Trans. on Computers, vol. C-32, no. 12,* December 1983.

[5] T. Gruning, U. Mahlstedt, and H. Koopmeiners, "DIATEST: A fast diagnostic test pattern generator for combinational circuits," in *Proc. Int'l Conf. on Computer-Aided Design, pp. 194-197,* 1991.

[6] I. Hamzaoglu and J. H. Patel, "New Techniques for Deterministic Test Pattern Generation," in *Proc. of VLSI Test Symposium, pp. 446-452,* 1998.

[7] N. Jha and S. Gupta, *Testing of Digital Systems*, Cambridge University Press, 2003.

[8] W. Kunz and D. K. Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems–Test, Verification, and Optimization," in *IEEE Trans. on Computer-Aided Design, vol. 13, no. 9, pp. 1143-1158* September 1994.

[9] A. Veneris and M. S. Abadir, "Design Rewiring Using ATPG," in *Proc. IEEE Trans. on Computer-Aided Design, vol. 21, no. 12, pp. 1469-1479,* December 2002.

Table 2: Fault Equivalence and DATPG (Step 2)

ckt	ATPG			DATPG			CPU (sec)	CPU comparison	
name	# pairs	# final	% err	# pairs	faults detected		fault	DATPG	DIATEST
	checked	classes	PVS	checked	one	both	equivalence	proposed	[7]
c432	1	419	0.2	111	70	41	0.03	0.00	0.00
c499	17	1106	1.3	84	28	56	0.14	0.02	0.08
c880	104	892	2.7	128	45	83	0.02	0.01	0.01
c1355	18	1094	1.4	80	25	55	0.22	0.07	0.05
c1908	335	830	7.6	129	39	90	0.06	0.05	0.01
c2670	6203	1443	18.0	127	73	54	0.07	0.05	0.05
c3540	8914	1839	14.1	116	62	54	0.10	0.09	0.09
c5315	184	3480	1.9	129	42	87	0.11	0.09	0.09
c6288	892	6973	5.4	94	77	17	0.42	0.05	0.09
c7552	8768	4737	9.8	130	46	84	0.50	0.16	0.13