On Applying the Set Covering Model to Reseeding

Silvia CHIUSANO, Stefano DI CARLO, Paolo PRINETTO
Politecnico di Torino, Dipartimento di Automatica e Informatica, Italy
{chiusano, dicarlo, prinetto }@polito.it
http://www.testgroup.polito.it

Hans-Joachim WUNDERLICH
Computer Architecture Lab, University of Stuttgart, Germany
wu@informatik.uni-stuttgart.de
http://www.ra.informatik.uni-stuttgart.de/

Abstract[1]

The Functional BIST approach is a rather new BIST technique based on exploiting embedded system functionality to generate deterministic test patterns during BIST. The approach takes advantages of two well-known testing techniques, the arithmetic BIST approach and the reseeding method.

The main contribution of the present paper consists in formulating the problem of an optimal reseeding computation as an instance of the set covering problem. The proposed approach guarantees high flexibility, is applicable to different functional modules, and, in general, provides a more efficient test set encoding then previous techniques. In addition, the approach shorts the computation time and allows to better exploiting the trade-off between area overhead and global test length as well as to deal with larger circuits.

1. Introduction

Recently a rather new BIST technique have been proposed, aiming at covering not random testable faults via deterministic test patterns generated through the available system modules. The basic idea of the approach can be summarized as follows: let two modules M_i and M_j be given, both part of the system mission logic and functionally connected. During testing, control M_i in such a way that its outputs are suitable test patterns for module M_j. M_i is typically a sequential circuit, and M_j is a combinational or pipelined unit.

The approach takes advantages of two well-known testing techniques, the *arithmetic BIST approach* [1][2] and the *reseeding method* [3][4]. From the former it derives the idea of exploiting the available system functionality for testing the system itself, and from the latter the idea of adequately initialize (*seed*) the test pattern generator unit to generate deterministic test sets.

The approach has been named *Functional BIST* [5][6][7][8], since it is not restricted to any specific modules M_i but it can work with any type of functions. The target application scenario is testing the actual System-On-Chips (SoCs), which include a variety of functional units, library modules (e.g., ALU, MAC, LFSR, etc.), as well as custom blocks. These modules usually form a strongly connected network, in which each unit is functionally linked to many other system modules either by bus- or by multiplexer-oriented interconnections.

Because of the novelty of the approach, so far a few papers addressed the problem of computing the appropriate initialization values (*reseedings*) for a given unit M_i used as test pattern generator (TPG). [5] proposes a simulation-based and an analytic method to compute the initialization values for an adder-based TPG. [6] still deals with adder based accumulator structures, and is able to compute seeds so that the resulting test sequences obtain complete fault coverage for all the ISCA'S85 circuits and the combinational parts of the ISCAS'89 circuits [9][10].

[7][8] present a *universal* algorithm called *GATSBY* (Genetic Algorithm based Test Synthesis tool for BIST applications), to compute the initialization values for a generic module used as TPG. Different test pattern generators were evaluated taking into account the parameters *test length*, *area overhead*, and *fault coverage*. Experiments show that GATSBY was able to outperform results presented into the literature and customized on specialized cases. However, since the GATSBY computation process strongly relies on simulation, the approach is not applicable to large circuits.

The goal of the present paper is to propose an effective method for reseeding computation. The approach guarantees the same flexibility of GATSBY but provides better reseeding solutions reducing the area overhead, and allows dealing with larger Unit Under Tests. The key point of the presented approach consists in formalizing a very innovative problem by resorting to the set covering

[1] This work was partially supported by *Deutscher Akademischer Austauschdienst* (DAAD) and by the *Conferenza dei Rettori delle Università Italiane* (CRUI), under the *Vigoni Project* 1999-2000.

techniques, which are well known in the computer design area and have been widely used in the past.

Set covering techniques have many applications in computer design, such as two-levels logic minimization, two-levels Boolean relation minimization, state minimization, exact encoding and DAG covering [11][12][13], and for optimal encoding of microprocessor instructions [14]. Moreover, they were exploited for test compaction in the testing field [15].

In the present paper, starting from an initial reseeding solution, a minimal one is computed by resorting to typical set covering techniques, based on essentiality and dominance [17], together with LINGO [16], a well-known and very efficient tool for solving linear programming problems. The present paper is organized as follows: Section 2 briefly overviews the basic concepts of Functional BIST approach; Section 3 details the formalization through the set covering model and describes the computation algorithm. Section 4 reports the experimental results, and Section 5 eventually draws some conclusions.

2. The Functional BIST: a summary

This section briefly summarizes the functional BIST approach and provides concepts and notations needed below. A deeper description of the approach is out of the scope is this paper, but the reader may refer to [7] and [8] for more details.

The aim of the approach is testing a given system through the functionalities available into the system itself. The modules used as a the *Test Pattern Generators* (TPGs) are generally sequential circuits, having input and internal state register partially or fully accessible, either via parallel load or in full-scan mode. To generate the appropriate test sequence, the TPG is first seeded by setting its *state register* and its *inputs register* to two initialization values, respectively δ and σ. Then, the TPG is let evolve for τ clock cycles. During the TPG evolution, the TPG input register remains σ, but the content of the state register is potentially updated at each clock cycle. The *Test Set* (TS) computed by the TPG is the sequence of τ patterns which appears on the TPG outputs, one pattern p_j at each clock cycle t_j, $0 \le j < \tau$: TS=$\{p_0, p_2, ...p_{\tau-1}\}$. The test set TS is characterized through the intrinsic functionality of the TPG itself as well as the triplet of values δ, σ, and τ employed to control the TPG evolution. Two elements mainly contribute in defining the quality of the test set: the *fault coverage* (FC%) and the *test length* (τ).

Experiments in [7] and in [8] showed that the complete fault coverage is not always achieved through a single triplet when dealing with large Unit Under Tests (UUTs). In this case, multiple TPG reseedings are required: periodically the TPG evolution has to be stopped and restarted with a new triplet, until the target fault coverage is reached.

A reseeding solution is a set of K triplets $\cup_{0 \le i <K} (\delta, \sigma, \tau)_i$, which are sequentially applied to the TPG. Each triplet drives the TPG evolution generating a test set TS_i which detects a percentage $\Delta FC\%_i$ of the UUT faults not covered by the other triplets. The overall test set TS is therefore the union of the test sets TS_i generated by each triplet: TS= $TS_0 \cup TS_1 \cup TS_2 \cup ... TS_{K-1}$; it is characterized by a global test length $T=\Sigma_{0 \le i < K} \tau_i$ and the fault coverage $FC\% = \Sigma_{0 \le i < K} \Delta FC\%_i$.

An optimal reseeding solution can be computed by trading-off the number of reseedings vs. area overhead and test length. *A low number of reseedings* allows minimizing the area needed to store the triplets (e.g., in a ROM), but usually a larger test length is necessary and the 100% of testable fault coverage is not always guaranteed. On the other hand, *large number of reseedings* guarantees the target fault coverage, with a shorter test length, but it implies more area overhead.

3. The Set Covering Model

The present paper addresses the computation of an optimal reseeding solution, which minimizes the number of reseedings. This value in fact strongly impacts on the applicability of the approach since it affects the area overhead.

Let $F=\{f_1, f_2, f_3,....\}$ be the target list of stuck-at faults of the combinational circuit to be tested. Our purpose is to compute a *minimal set of triplets* $\cup(\delta, \sigma, \tau)_i$ such as the resulting test set TS= $\cup TS_i$ guarantees the detection of all the faults belonging to F. This problem can be formalized as an instance of the *set covering problem*. In the following, for sake of readiness, the triplet of values $(\delta, \sigma, \tau)_i$ will be denoted as $triplet_i$.

Let us start with a initial reseeding of M triplets $T=\{triplet_0, triplet_1, \ldots, triplet_{M-1}\}$, built up in order to guarantee the detection of all the target faults $F=\{f_1, f_2, f_3,....\}$. By construction $F= \cup_{tripleti \in T} F(triplet_i)$, being $F(triplet_i) = \{f_{1/i}, f_{2/i}, f_{3/i},....\}$ the subset of faults detected by the test set TS_i, generated by $triplet_i$. The object of the research becomes find a set N of triplets, $N \subseteq T$, such that $\cup_{tripleti \in N} F(triplet_i) = F$ and N has minimum cardinality.

Definition: A set N of triplets is a minimal solution iff none its triplet can be removed without affecting the detection of F.

Therefore, each $triplet_i \in N$ is necessary to detect at least one fault $f_k \in F$, which is not covered by any other triplet of N.

To map this problem as an instance of the set covering problem, let define a matrix, named in this context *Detection Matrix*, having size (#Triplets$\in N$)×(#Faults$\in F$). Each row of the matrix corresponds to a $triplet_i \in T$ and each column to a fault $f_j \in F$. Each cell d_{ij} of the matrix is

set to '1' if at least one pattern of the test set TS_i, generated by $triplet_i$, detects the fault f_j; it is fixed to '0',otherwise.

Further, define a vector x of M Boolean variables, such that x_j provided by $triplet_j$ is selected for inclusion in N. Our goal is therefore to solve the following integer optimization problem:

minimize $\Sigma_j x_j$

constraints *Detection Matrix* $\bullet\ x \geq 1$, $x \in \{0, 1\}^M$

which can also be viewed as an instance of the set covering problem.

The quality of the final solution N strongly depends on both the goodness of the initial solution T as well as on the adopted set covering algorithm. In the following three Sections, these aspects will be analyzed in detail.

3.1. Building up the Detection Matrix

The initial reseeding T is generated by resorting to the test set *ATPGTS* provided by a commercial gate-level ATPG tool, which guarantees complete covering of F. The cardinality of T is fixed equal to the test length of *ATPGTS*.

Being *ATPGTS* a sequence of M patterns p_i, *ATPGTS* $=\{p_0, p_2, .., p_{M-1}\}$, for each $triplet_j$ of T the value δ is set to one pattern p_j, and the value σ is randomly selected. The number τ of clock cycles for triplet evolution is experimentally tuned and applied to all the triplets of T. Fixing τ='0', the test set *TS* provided by the reseeding corresponds to the ATPG test set *ATPGTS*.

3.2. Detection Matrix Reduction

First, the Detection Matrix is simplified using *essentiality* and *dominance* methods [17]. The two techniques are iteratively applied until the matrix cannot be reduced any more.

Definition: $triplet_i$ is essential or *necessary* iff at least one fault $f_j \in F$ is detected only by one of the pattern belonging to TS_i.

Necessary triplets must be included into the final solution N. The Detection Matrix is therefore simplified deleting all the rows corresponding to the necessary triplets, and all the columns corresponding to the faults detected by them ($F(triplet_i)$).

Definition: $triplet_j$ is dominated by $triplet_k$ iff $F(triplet_j) \subseteq F(triplet_K)$, i.e., the set TS_k detects the faults covered by TS_j plus possibly some additional others.

Dominated triplets will not be included into the final reseeding solution N and therefore the corresponding rows are removed from the Detection Matrix.

3.3. Computing an Optimal Reseeding Solution

If the Detection Matrix is empty at the end of the reduction process, the final reseeding solution N will only contain necessary triplets. Otherwise the reduced matrix must be further analyzed by resorting to alternative solving algorithms. Depending on the size of the matrix, either exact approaches or local research and meta-heuristic techniques are applied. Experiments reported in Section 4 show that on this kind of problems the reduction process is highly effective, and the size of the reduced matrix allow to deal it with an exact algorithm. In particular, among the algorithms available into the literature, we decided to adopt the linear programming software package LINGO, an effective commercial tool that addresses the exact solution of combinational optimization problems [16].

4. Experimental results

Figure 1 sketches the overall computation flow of the proposed set covering based method. First, the *Initial Reseeding Builder* provides the starting reseeding solution (T) and computes the Detection Matrix. It receives as an input the behavioral description of the TPG, together with the *ATPGTS* deterministic test set and the fault list F, both provided by a gate-level ATPG. Then, the *Matrix Reducer* simplifies the Detection Matrix and computes the set of necessary triplets. Finally, the software package *LINGO* [16] post processes the matrix, extracting a minimal subset of triplets. The computed reseeding solution (N) is therefore the union of the *necessary triplets* and the *minimal subset of triplets*.

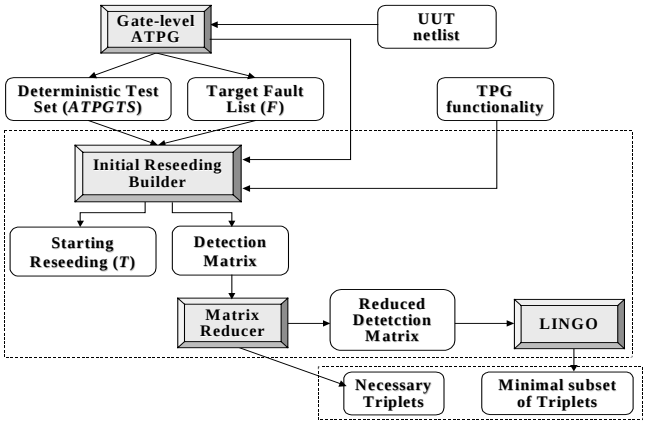

Figure 1: The reseeding computation flow

To run the experiments, in the present paper both the *Initial Reseeding Builder* and *Matrix Reducer* have been implemented in ANSI C. The target fault list (F) and the *ATPGTS* are instead computed by resorting to the gate-level ATPG TestGen [18]. The tool is also employed to support the computation of the initial reseeding solution (T) and the Detection Matrix. The *Initial Reseeding Builder* builds up a $triplet_i$ for each pattern p_i belonging to *ATPGTS* and computes the corresponding test set TS_i, seeding the TPG by $triplet_i$ and let it evolving for τ_i clock cycles. The fault coverage ΔFC_i for TS_i is gathered fault

simulating TS_i on *F* through the TestGen fault simulator. To build the matrix, the value τ_i is experimentally tuned and fixed equal for all the triplets of *T*.

As TPGs, we focus on three accumulator-based units including arithmetic functions such as *adder*, *multiplier* and *subtracter*, which are quite common in the actual SoCs. As UUT we consider the ISCAS'85 and the full-scan version of ISCAS'89 benchmark circuits [9][10], which are not random testable by 10k patterns. Final reseeding solutions are collected in Table 1, whereas Table 2 and Figure 2 allow a deeper analysis of the results. Experiments were run on a Sun SparcStation 5/110 with 64Megabytes of RAM.

Table 1 reports the cardinality of the reseeding solution (*#Triplets*) and the global test length (*Test Length*) for each considered TPG. Moreover Table 1 compares the actual results and the GATSBY solutions [8]. On all the circuits (except s838) the set covering based approach sensibly reduces the number of reseedings with respect to GATSBY: the improvement ranks from –2 to –25 triplets and interests all the three considered TPGs. Therefore, without loosing generality, the approach proposed in this paper provides solution significantly less costly in terms of area overhead to store the triplets. No comparison is available for s13207 and s15850 since the two circuits were too large to be dealt with by GATSBY.

One of the advantages of the set covering based approach rely on the fact that it shorts the computation time, allowing to better exploit the trade-off between the number of reseedings and the test length, possibly dealing with larger test sequences. W.r.t. GATSBY, the number of fault simulations is reduced and limited to the construction of the Detection Matrix.

In the case of multiple reseedings, the global test length reported in Table 1 is computed deleting from each test set TS_i the last subsequence of patterns not contributing to the fault coverage ΔFC_i. For each triplet therefore we assume to store both the seeding values δ and σ, and the actual number of clock cycles for the evolution. The area overhead can be further reduced let evolving all the triplets for the same interval of time. In this case the value τ must be the largest number of clock cycles among the ones required by each triplet of the reseeding solution.

Figure 2 focuses on the trade-off between the number of reseedings and the test length, in the case of the circuit s1238 and considering as TPG an adder based accumulator. Starting from a test length of 5,427 and progressively increasing this value to 15,551, the number of triplets decreases from 11 to 2.

Table 2 focuses on the complexity of the problem, and on the characteristics of the reseeding solutions. The first column of Table 2 reports the size of the initial Detection Matrix, expressed as #Triplets×#Faults. By construction #Triplets is the test length of the TestGen test set. The remaining columns show, for each TPG, the impact of the reduction techniques and the contribution of LINGO. Experiments show that the reduction is quite effective on this kind of problems, allowing to significantly prune the Detection Matrix and providing a matrix that can be processed by LINGO. On some examples (c499, c880, c1355, c1908, s820, s838, s953, s1423, s15850) the reseeding solution only contains necessary triplets, being the matrix empty after reduction. On the others, the reseeding includes either no necessary triplets (s420, s641, s1238, s5378, s9234, s13207) or both necessary triplets and triplets computed by LINGO (c7552, s9234).

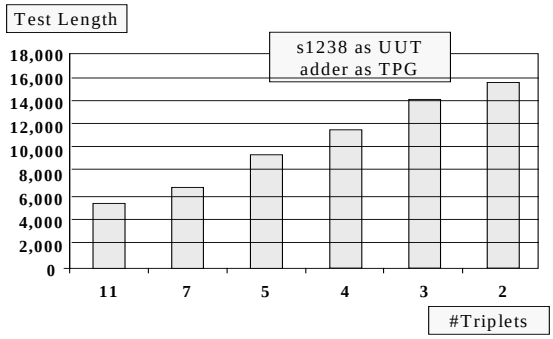

Figure 2: Trade-off Reseedings vs.Test Length

5. Conclusions

The present paper works in the area of the Functional BIST and proposes an effective method, based on set covering techniques, for optimal reseeding computation. Experiments show that the approach allows conjugating effectiveness and high flexibility. On one hand, it is not customized on specific test pattern generators. On the other hand, it allows exploiting the trade-off between area overhead and global test length, and it provides reseeding solutions with minimum area overhead.

6. Acknowledgments

The authors wish to thank Alberto Olivero for implementing the algorithm and performing the experiments, and Federico Della Croce for the fruitful discussions.

7. References

[1] J. Rajski, J. Tyszer, *Arithmetic Built-In Self-Test for Embedded Systems*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998

[2] S. Gupta, J. Rajski, J. Tyszer, *Arithmetic Adaptive Generators of Pseudo-Exhaustive Test Patterns*, IEEE Trans. on Computers, 8(45): 939-949, August, 1996

[3] S. Hellebrand, S. Tarnick, J. Rajski, B. Courtois, *Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers*, IEEE ITC, 1992, pp. 120-129

[4] S. Hellebrand, B. Reeb, S. Tarnick, H.-J. Wunderlich, *Pattern Generation for a Deterministic BIST Scheme*, IEEE ICCAD, 1995, pp. 88-94

[5] A. P. Stroele, F. Mayer, *Methods to reduce Test Application Time for Accumulator-Based Self –Test*, IEEE VTS, 1997, pp. 48-53

[6] R. Dorsch, H.-J. Wunderlich, *Accumulator Based Deterministic BIST*, IEEE ITC, 1998

[7] S. Cataldo, S. Chiusano, P. Prinetto, H-J. Wunderlich, *Optimal Hardware Pattern Generation for Functional BIST*, IEEE DATE, 2000, pp.292-297

[8] S. Chiusano, P. Prinetto, H-J. Wunderlich, *Non Intrusive BIST for System-on-a-Chip*, IEEE ITC, 2000

[9] F. Brglez and H. Fujiwara, *A Neutral Netlist of 10 Combinatorial Benchmark Circuits*, IEEE Int. Symp. on Circuits and Systems, 1985

[10] F. Brglez, D. Bryan, K. Kozminski, *Combinatorial Profiles of Sequential Benchmark Circuits*, IEEE Int. Symp. on Circuits and Systems, 1989

[11] M. Karnaugh, *The Map Method for Synthesis of Combinational Logic Circuits*, Transaction IEEE, vol. 72, pp 593-599, 1953

[12] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Acad. Pub., 1984

[13] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994

[14] T. Agrawala, *Microprogram Optimization: A Survey*, IEEE Trans. Comp., pp. 962-973, October 1976

[15] I. Pomeranz, L. N. Reddy, S. M. Reddy, *COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits*, IEEE Trans. on CAD, pp. 1040-1049, July 1993

[16] *LINGO 5.0 User Manual*, LINDO System INC, 1999

[17] E. L. Jr. McCluskey, *Minimization of Boolean Functions*, Bell. Sys. Tech. Jour., vol. 35, pp. 1417-1444, April 1959

[18] http://www.synopsys.com/

	Set Covering						Set Covering -GATSBY					
	Adder		Multiplier		Subtracter		Adder		Multiplier		Subtracter	
Circuit	# Triplets	Test Length	# Triplets	Test Length	# Triplets	Test Length	Δ #Triplets	ΔTest Length	Δ #Triplets	ΔTest Length	Δ #Triplets	ΔTest Length
c499	1	650	1	690	1	484	0	282	0	336	0	64
c880	1	3935	1	3,122	1	3,686	0	1,831	0	1,156	0	1563
c1355	1	1816	1	1,796	1	1,708	0	665	0	555	0	531
c1908	1	3845	1	3,807	1	3,929	0	72	0	459	0	288
c2670	18	168,072	21	224,048	19	221,970	-15	157,893	-15	215,738	-11	211,569
c7552	38	286,725	38	286,200	39	297,579	-26	236,725	-32	246,200	-29	247,579
s420	4	111,899	5	106,132	5	136,769	-3	106,389	-5	96,421	-5	127,918
s641	3	69,473	4	78,649	4	97,005	-2	64,998	-2	75,919	-3	94,340
s820	1	7,075	1	13,410	1	8,219	-2	1,764	-2	8,009	-2	734
s838	70	833,217	67	769,309	72	897,366	59	826,523	-25	758,321	42	880,174
s953	1	16,855	1	19,053	1	10,910	-2	8,984	-2	13,250	-3	4,338
s1238	2	15,551	2	26,407	2	15,704	-2	8,195	-4	17,336	-5	7,172
s1423	1	6,916	1	13,954	1	13,707	-2	3,816	-3	9,480	-2	10,268
s5378	3	22,848	3	23,476	3	23,636	-4	12,848	-6	11,476	-3	12,636
s9234	46	182,100	39	224,770	36	210969	-4	132,100	-16	164,770	-17	155,969
s13207	15	36,102	18	42,130	17	40,610	-	-	-	-	-	-
s15850	104	208,236	92	362,632	83	328,100	-	-	-	-	-	-

Table 1: Reseeding solution

		Adder			Multiplier			Subtracter		
		Reduction			Reduction			Reduction		
Circuit	Initial Detection Matrix	Reduced Detection Matrix	Necessary Triplets	LINGO	Reduced Detection Matrix	Necessary Triplets	LINGO	Reduced Detection Matrix	Necessary Triplets	LINGO
c499	74X692	0X0	1	-	0X0	1	-	0X0	1	-
c880	91X850	0X0	1	-	0X0	1	-	0X0	1	-
c1355	118X1,508	0X0	1	-	0X0	1	-	0X0	1	-
c1908	184X1,816	0X0	1	-	0X0	1	-	0X0	1	-
c2670	163X2,635	103X307	0	18	91X196	0	21	106X344	1	18
c7552	326X7,396	28X32	29	9	0X0	38	-	0X0	39	-
s420	110X421	98X125	0	4	103X97	0	4	93X231	0	5
s641	92X463	14X10	0	3	11X10	0	4	12X112	0	4
s820	176X806	0X0	1	-	0X0	1	-	0X0	1	-
s838	215X865	0X0	70	-	0X0	67	-	0X0	72	-
s953	114X995	0X0	1	-	0X0	1	-	0X0	0	2
s1238	236X1,311	34X239	0	2	49X329	0	2	185X134	0	2
s1423	93X1,391	0X0	1	-	0X0	1	-	0X0	1	-
s5378	354X4,195	354X609	0	3	354X718	0	3	354X260	0	3
s9234	573X6,613	512X308	11	35	476X226	12	27	567X432	5	31
s13207	623X8,991	621X3,353	0	15	623X3,520	0	18	623X4,063	0	17
s15850	626X10,729	0X0	104	-	0X0	92	-	0X0	83	-

Table 2: Set Covering algorithm