Dynamic Compaction in SAT-Based ATPG*

Alexander Czutro[1], Ilia Polian[1], Piet Engelke[1], Sudhakar M. Reddy[2], Bernd Becker[1]

[1]Institute for Computer Science
Albert-Ludwigs-University
D-79110 Freiburg i. Br., Germany

[2]ECE Department
University of Iowa
Iowa City, IA 52242, USA

Abstract

SAT-based automatic test pattern generation has several advantages compared to conventional structural procedures, yet often yields too large test sets. We present a dynamic compaction procedure for SAT-based ATPG which utilizes internal data structures of the SAT solver to extract essential fault detection conditions and to generate patterns which cover multiple faults. We complement this technique by a state-of-the-art forward-looking reverse-order simulation procedure. Experimental results obtained for an industrial benchmark circuit suite show that the new method outperforms earlier static approaches by approximately 23%.

Keywords: SAT-based ATPG, Dynamic compaction

1 Introduction

Recent advances in Boolean-satisfiability (SAT) solvers are increasingly rendering SAT-based automatic test pattern generation (ATPG) [10,15] an attractive alternative to traditional structural approaches [6, 7, 9, 14]. Particular strengths of SAT-based ATPG algorithms are the ability to resolve hard-to-detect faults in large industrial circuits [2, 3] and to produce a quick redundancy proof for untestable faults. Furthermore, SAT-based ATPG approaches are easily applicable to non-standard fault models such as resistive bridging faults, which require non-trivial constraints for fault activation [2].

One weakness of SAT-based ATPG methods is their relatively high pattern count, which is due to overspecification of the calculated test patterns. Structural test generators typically search for a pattern starting at the fault site and moving towards the circuit's inputs. In this way, only inputs necessary for fault detection tend to be assigned 0 or 1 values; the remaining inputs have don't-care values which can subsequently be assigned to detect other faults, thus reducing the size of the test set. A SAT-based ATPG searches for a test pattern based on the SAT engine's decision heuristics, which may or may not reflect the circuit's structure. Hence, the generated patterns have only few don't-care positions.

*Parts of this work have been supported by the German Research Council project RealTest (BE 1176/15-2) and by the Alexander-von-Humboldt Foundation. We are thankful to Juergen Schloeffel of NXP / Mentor Graphics Hamburg for providing industrial circuits.

Test pattern compaction methods in structural test generation can be divided in two classes: static and dynamic compaction [1]. Static compaction starts with a generated test set and produces a smaller test set which detects (at least) the same faults as the original test set. This is done by merging compatible patterns or by eliminating patterns which do not detect any fault not covered by other patterns. Dynamic compaction considers test set minimization during the test generation process by generating test patterns which detect multiple faults [8, 12].

For SAT-based ATPG, static compaction based on merging is challenging because the generated patterns have few don't-cares. A simple static compaction method based on injecting don't-cares on inputs outside the input cone of the fault site and its propagation path to the output was proposed in [5]. Although high don't-care densities are reported for industrial circuits, the efficiency of this technique may not be high, as faults from the same cone are still likely to result in test patterns with conflicting assignments to inputs. Even though don't-cares could be injected on these inputs without sacrificing detectability, the method from [5] does not handle such situations. Pattern counts after, but not before compaction are reported, so the practical efficiency of the technique in reducing the test set size is unknown.

In this paper, we introduce a dynamic compaction procedure for SAT-based ATPG. It targets fault groups which are enlarged consecutively. Whenever a pattern is generated for fault f, necessary assignments to the circuit lines are extracted from the ATPG's internal data structures. The procedure then attempts to add another fault (say, f_n) to the current group (say, $f_1, f_2, \ldots, f_{n-1}$) by generating a test for f_n while enforcing that the assignments for detecting fault f_1 through f_{n-1} are not violated. If test generation is successful, the pattern is guaranteed to detect all the faults $f_1, f_2, \ldots, f_{n-1}$ as well as the new fault f_n. The approach is different from classical dynamic compaction for structural ATPG which fixes inputs of the circuit rather than internal circuit lines after adding a new fault to a group.

First experimental results obtained by our ATPG tool TIGUAN (Thread-parallel Integrated test pattern Generator Utilizing satisfiability ANalysis) [2] show an average reduction in pattern count by approximately one half compared to the standard ATPG flow with fault dropping. Compared

1081-7735/09 $26.00 © 2009 IEEE
DOI 10.1109/ATS.2009.31

IEEE computer society

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on June 09,2010 at 09:22:04 UTC from IEEE Xplore. Restrictions apply.

to cone-based don't-care injection followed by static compaction (which can be regarded as our implementation of [5]), the dynamic compaction procedure achieves 23% reduction in pattern count and a slight improvement in overall run time.

The remainder of the paper is organized as follows. SAT-based ATPG is reviewed in Section 2, where emphasis is put on data structures used for dynamic compaction. The dynamic compaction procedure is introduced in Section 3. In Section 4, experimental results on an industrial benchmark suite are reported. Section 5 concludes the paper.

2 SAT-Based Test Pattern Generation

A SAT-based ATPG tool such as PASSAT [3] or TIGUAN [2] takes a circuit, a fault list and, possibly, further parameters such as the timeout (time budget which the tool is allowed to spend to classify a fault as either detected or undetected). When no dynamic compaction is active, the tool selects one fault from the fault list and attempts to generate a test pattern for this fault by constructing the *conjunctive normal form* (CNF) of the miter circuit [10, 15] (see Section 2.2). The CNF is handed to a Boolean SAT solver (PASSAT calls MiniSAT [4] while TIGUAN is tightly integrated with the thread-parallel solver MiraXT [11]). If the solver finds a model of the instance, the fault is detectable and the pattern is derived from the model. An unsolvable SAT instance is formal evidence of a redundant fault. A fault which caused the solver to time out is considered aborted.

Once all faults have been classified as detected, redundant or aborted, static compaction by pattern merging can be performed. TIGUAN injects don't-cares into test patterns based on the technique from [5] and runs the standard greedy merging algorithm [1]. After that, the remaining don't-cares are randomly filled with specified values and reverse-order simulation is performed: patterns are simulated in reverse order, and patterns which do not detect new faults are eliminated.

In the following, we describe the conditional multiple stuck-at (CMS@) fault model used by TIGUAN (Section 2.1) and provide some details on CNF generation (Section 2.2). The CMS@ fault model was introduced in [2] as a generic interface to handle non-standard defect classes. Although only stuck-at faults are considered in this paper, the CMS@ fault model is utilized by dynamic compaction to constrain the ATPG process such that the generated pattern detects multiple faults. Auxiliary data structures called D-chains [15] provide the source of the constraints for detecting multiple faults and are explained in Section 2.2.

2.1 CMS@ Fault Model

A CMS@ fault consists of two lists of length r and s, respectively. The *list of aggressor lines* has the form $\{a_1/a_1^{val}, \ldots, a_r/a_r^{val}\}$. The *list of victim lines* is of the form $\{v_1/v_1^{val}, \ldots, v_s/v_s^{val}\}$. a_i and v_j denote signal lines, while all a_i^{val} and v_j^{val} indicate a logical value (0 or 1).

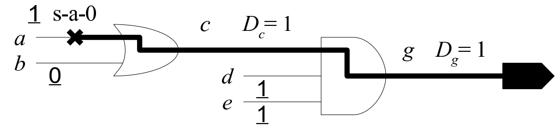

Figure 1: D-chain assignments

The behavior of a circuit under a CMS@ fault is as follows. If an input vector sets every aggressor line a_i to value a_i^{val}, the fault is activated and the value on each victim line v_j changes to v_j^{val}. Otherwise, no fault effect is present. A v-single-stuck-at-val fault is represented by an empty aggressor list and victim list $\{v/val\}$. Representation of further fault models by CMS@ faults is explained in [2].

2.2 CNF generation

The SAT instance to generate a test pattern for CMS@ fault $f = \{a_1/a_1^{val}, \ldots, a_r/a_r^{val}\}, \{v_1/v_1^{val}, \ldots, v_s/v_s^{val}\}$ in a circuit C models a miter circuit of the sub-circuit C_r of C enriched by auxiliary structures called D-chains. Circuit C_r contains the parts of C which are relevant for fault detection.

Every line i in circuit C_r is assigned two Boolean variables: G_i (value of the line in the fault-free (good) circuit) and B_i (value of the line in the faulty (bad) circuit). For a line i in the output cone of the victim line(s) of the fault, a third variable $D_i \equiv (G_i \oplus B_i)$ is introduced. D_i is 1 if line i is sensitized to the fault. We require that $D_i = 1$ implies that the D variable of at least one input of the gate driving i must also be 1 (this condition is not enforced at the fault site).

A successful test generation produces a sensitized path to an output marked by D variables set to logic-1 (D-chain). The D-chain technique was introduced to speed up test generation by guiding the search (the existence of a test pattern does not depend on the D variables) [15]. In this work, we utilize the information on the sensitized paths to generate patterns for multiple faults. Figure 1 illustrates a D-chain.

3 Dynamic Compaction Procedure

The dynamic compaction procedure is outlined in Figure 2. It attempts to find a *fault group*, i.e., a collection of multiple faults for which a common test pattern exists, and to generate that pattern. The procedure first sorts the fault list topologically (Line 1). Fault group construction starts with one fault. In every iteration (Lines 3–22), the algorithm attempts to add one more fault to the fault group by trying to generate a pattern which detects the new fault while still detecting the faults already contained in the fault group. Actual test generation is done by procedure **test_gen** (Line 7), which takes two arguments: fault f and a list of additional conditions FGA (fault group assignments) and returns the pattern if the test generation is successful and the distinguished return value $\perp$ otherwise.

In the beginning, FGA is empty and **test_gen** performs normal single-stuck-at test generation: the CNF is constructed and TIGUAN is invoked for the target stuck-at fault f. If no pattern is found for the single fault f, this fault is

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on June 09,2010 at 09:22:04 UTC from IEEE Xplore. Restrictions apply.

```
Procedure dynamic_compaction
Input: Fault list F
Output: Compact test set T
(1)  Sort the fault list;
(2)  T := ∅; FGA := ∅; t_old := ⊥;
(3)  repeat
(4)      if (F empty) then t := ⊥;
(5)      else begin
(6)          f := first fault from F;
(7)          t := test_gen(f, FGA);
(8)      end if
         // Case 1: Redundant or aborted fault
(9)      if (t == ⊥ and t_old == ⊥)
(10)     then eliminate f from F;
         // Case 2: f conflicts with current fault group
(11)     else if (t == ⊥ and t_old ≠ ⊥)
(12)     then begin
(13)         Add t_old to T;
(14)         Fault simulate t_old, eliminate det. faults from F;
(15)         FGA := ∅; t_old := ⊥;
(16)     end if
         // Case 3: Current fault group successfully enlarged
(17)     else begin
(18)         Extract detection conditions from D-chains
               and add them to FGA;
(19)         t_old := t;
(20)         Eliminate f from F;
(21)     end if
(22) until (F empty);
(23) return T;
end dynamic_compaction;
```

Figure 2: Dynamic compaction algorithm

removed from the fault list and the next fault from F is tried as the first fault of a fault group in the next iteration (Lines 9–10). If test generation is successful (Lines 17–21), the dynamic compaction procedure extracts a compact set of value assignments which are sufficient to detect the fault. These assignments are:

- For the stuck-at-val fault at line v, v is assigned $\overline{val}$.
- From all gates with the D variable at their output set to logic-1, the gates which form one path to an output of the circuit are selected. All side-inputs of these gates are assigned the gate's non-controlling value.

In Figure 1, $D_c = D_g = 1$ indicate the propagation path to detect the a-stuck-at-0 fault. The sufficient assignments to detect this fault are 1 at a (for fault activation), 0 at b and 1 at d and e (for propagation), written as $\{a/1, b/0, d/1, e/1\}$. Every test pattern which sets lines a, b, d and e according to these assignments, will detect the a-stuck-at-0 fault.

In Line 18, the extracted assignments are added to the fault group assignment set FGA which is empty in the beginning. As soon as a pattern is found for a fault, sufficient assignments to detect this fault are contained in FGA in the next iteration. Generating a test pattern for a fault f' by **test_gen**(f', FGA) ensures that, if a pattern is generated, the values of all circuit lines under this pattern are consistent with the assignments in FGA. Technically, this is implemented by adding FGA as the aggressor list to the CMS@ fault while keeping the victim list from fault f'. Suppose that in the example in Figure 1, f' is the stuck-at-1 fault at a line h elsewhere in the circuit. Adding fault f' to FGA is done by targeting the CMS@ fault $\{a/1, b/0, d/1, e/1\}, \{h/1\}$. (If a pattern is found, it still detects the a-stuck-at-0 fault.)

If fault f' is targeted in the second iteration and the test generation is successful, the generated pattern detects all faults in fault group $\{f, f'\}$. New assignments are extracted from the D-chain information and added to FGA. Every pattern generated with the new FGA as the second input to procedure **test_gen** is guaranteed to still detect f and f'.

In every iteration, another fault is added to the fault group, until test generation fails, indicated by $t = \perp$ (Lines 11–16). This can be due to incompatibility of the detection conditions for the new fault to assignments collected in FGA so far (Line 7) or because all faults have been detected (Line 4). The last successfully generated test pattern, t_{old}, is included into the test set T, and fault dropping is performed (Line 14). The current fault is used as the start for a new fault group.

The procedure terminates when no undetected faults are left. The procedure returns the calculated test set T.

4 Experimental Results

We integrated the dynamic compaction procedure into our ATPG tool TIGUAN and applied it to the combinational cores of industrial circuits provided by NXP. Although the same benchmark suite was used in [5], the circuits appear to be synthesized using different options and have a different structure, as indicated by the discrepancy in gate counts. We found that the results of our re-implementation of [5] and the results published in [5] did not track well. Therefore, when we quote results on static compaction, we refer to our own implementation. We used a timeout of 1 second per fault and no multithreading.

Table 1 compares the pattern counts achieved by TIGUAN with fault dropping and no further compaction, TIGUAN followed by don't-care injection and static compaction, and TIGUAN with dynamic compaction. For each of these alternatives, the pattern counts before and after reverse-order simulation are indicated by BROS and AROS, respectively. In addition, we implemented forward-looking reverse-order simulation: for every fault, the first test pattern which detects it is stored; during the reverse-order simulation, all the patterns which are not the first detecting patterns for any faults are dropped without simulation [13]. The pattern counts obtained by replacing reverse-order fault simulation by forward-looking reverse-order fault simulation are denoted by AFLROS. For all three scenarios, forward-looking reverse-order simulation outperforms reverse-order simulation for every circuit. The final row of Table 1 contains the cumulated numbers for all the benchmarks.

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on June 09,2010 at 09:22:04 UTC from IEEE Xplore. Restrictions apply.

Table 1: Pattern count for different compaction methods before (BROS) and after reverse-order simulation (AROS) and after forward-looking reverse-order fault simulation (AFLROS)

		No compaction			Static compaction			Dynamic compaction			Run-time [s]		
Circuit	Gates	BROS	AROS	AFLROS	BROS	AROS	AFLROS	BROS	AROS	AFLROS	No comp.	Static	Dynamic
p35k	48927	9896	7196	6705	9414	8957	6305	5225	4655	4151	3049.0	4893.8	5339.5
p45k	46075	3450	2896	2747	3257	3248	3108	3204	3138	3078	241.8	629.2	906.8
p77k	75033	5293	3848	3231	2215	2149	1594	1609	1561	1373	10772.5	11044.3	10514.2
p78k	80875	1239	546	401	292	291	126	246	246	103	722.0	1494.0	3013.8
p81k	96722	8356	6509	5499	16844	16386	4392	4906	4860	1545	1765.9	11052.7	3177.3
p89k	92706	9141	6704	5847	7737	7706	4529	5861	5527	3912	2453.2	4956.5	4706.1
p100k	102443	5066	3645	3032	3113	3106	2801	3096	3093	2731	976.2	1867.0	3584.4
p141k	185360	8585	5845	5298	24717	22640	3952	9788	8942	3438	6894.3	24188.2	23673.0
p267k	296404	10222	8374	7441	7431	7103	3643	4475	4413	2879	5177.8	14334.9	17637.9
p269k	297497	10367	8452	7456	7465	7127	3638	4419	4367	2840	5940.1	13527.4	15019.9
p286k	373221	18738	14062	12240	22997	22501	7859	13212	12590	6407	14099.2	52224.3	48630.3
p295k	311901	17768	14636	12850	10406	10061	5017	11401	11132	6049	9418.0	47562.6	39785.4
p330k	365492	23546	18618	17676	16497	15874	12839	7252	7236	6547	12236.5	33720.7	27047.9
p378k	404367	3397	902	696	290	289	161	242	242	130	9251.0	18807.8	47684.5
p388k	506034	12198	8764	7684	4959	4939	2846	4883	4846	2813	12446.7	32830.6	44225.0
p469k	49771	610	430	341	1141	1061	352	739	647	330	27714.0	28286.9	41088.1
Sum	**3332828**	**147872**	**111427**	**99144**	**138775**	**133438**	**63162**	**80558**	**77495**	**48326**	**123158.2**	**301420.9**	**336034.1**

Both static and dynamic compaction are effective in reducing the test set size for all but one circuit (p45k). The total pattern count is reduced by 36% and 51% by these respective techniques. Dynamic compaction outperforms its static counterpart for 15 out of 16 circuits, reducing the total pattern count by 23%. (For reverse-order simulation or in absence of any post-processing, the improvements are around 42%.) Reverse-order and forward-looking reverse-order simulation are less effective after dynamic compaction than after static compaction because test patterns generated by dynamic compaction typically target multiple faults and are not very likely to cover no faults undetected by other patterns.

The final three columns compare the run-times needed by the three experiments. All measurements were performed on a 2.3 GHz AMD Opteron computer with 64 GB RAM (TIGUAN is a 32-bit application which does not consume any memory exceeding 4 GB). The run times consumed by both compaction types are similar for many circuits, although sometimes one method takes considerably longer than its counterpart.

5 Conclusions

We presented a dynamic compaction procedure for SAT-based ATPG which scales to multi-million gate designs and outperforms the static compaction method introduced before. The results demonstrate that dynamic compaction compensates for one drawback of SAT-based ATPG, namely the high pattern count. The procedure is tightly integrated into the ATPG tool and makes use of its specific data structures and interfaces.

Additional compaction can be achieved by better selection of the fault groups. We currently stop the fault group construction as soon as one fault turns out to be incompatible with the remainder of the group. It is possible to attempt adding other fault to the fault group, thus yielding patterns which detect even more faults.

6 References

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

[2] A. Czutro, I. Polian, M. Lewis, P. Engelke, S. M. Reddy, and B. Becker. TIGUAN: Thread-parallel Integrated test pattern Generator Utilizing satisfiability ANalysis. In *Int'l Conf. on VLSI Design*, pages 227–232, 2009.

[3] R. Drechsler, S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schlöffel, and D. Tille. On Acceleration of SAT-based ATPG for Industrial Designs. *IEEE Trans. on CAD*, 27(7):1329–1333, 2008.

[4] N. Eén and N. Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 541–638. Springer, 2003.

[5] S. Eggersglüß and R. Drechsler. Improving test pattern compactness in sat-based atpg. In *Asian Test Symp.*, pages 445–452, 2007.

[6] H. Fujiwara. FAN: A Fanout-Oriented Test Pattern Generation Algorithm. In *IEEE Int'l Symp. on Circuits and Systems*, pages 671–674, 1985.

[7] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Trans. on CAD*, 30:215–222, 1981.

[8] P. Goel and B. C. Rosales. Test generation and dynamic compaction of tests. In *Int'l Test Conf.*, pages 189–192, 1979.

[9] I. Hamzaoglu and J. H. Patel. New Techniques for Deterministic Test Pattern Generation. *Jour. Electronic Testing: Theory and Applications*, 15:63–73, 1999.

[10] T. Larrabee. Efficient Generation of Test Patterns Using Boolean Difference. In *Int'l Test Conf.*, pages 795–801, 1989.

[11] M. Lewis, T. Schubert, and B. Becker. Multithreaded SAT Solving. In *ASP Design Automation Conf.*, Yokohama, Japan, January 2007.

[12] I. Pomeranz, L. N. Reddy, and S. M. Reddy. COMPACTEST: A method to generate compact test sets for combinational circuits. In *Int'l Test Conf.*, pages 194–203, 1991.

[13] I. Pomeranz and S. M. Reddy. Forward-looking fault simulation for improved static compaction. *IEEE Trans. on CAD*, 20(10):1262–1265, 2001.

[14] J. P. Roth. Diagnosis of Automata Failures: A Calculus and a Method. *IBM Jour. of Research and Development*, 10:278–281, 1966.

[15] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Combinational Test Generation Using Satisfiability. *IEEE Trans. on CAD*, 15(9):1167–1176, September 1996.

Authorized licensed use limited to: UNIVERSITAET FREIBURG. Downloaded on June 09,2010 at 09:22:04 UTC from IEEE Xplore. Restrictions apply.