Journal on Satisfiability, Boolean Modeling and Computation 8 (2012) 83-88

TG-Pro: A SAT-based ATPG System

SYSTEM DESCRIPTION

Huan Chen huan.chen@ucd.ie
Joao Marques-Silva jpms@ucd.ie
CASL/CSI, University College Dublin
Dublin
Ireland

Abstract

Automatic Test Pattern Generation (ATPG) is arguably one of the practical applications that motivated the development of modern Boolean Satisfiability (SAT) solvers in the mid 90s. Despite the interest of using SAT in ATPG, the original model remained mostly unchanged for nearly two decades, even in the presence of renewed interest in applying modern SAT technology to large-scale hardware designs. This paper describes the SAT-based ATPG system TG-Pro. In contrast to all SAT-based ATPG work over the last two decades, TG-Pro is based on a new fundamentally different SAT-based ATPG model. Experimental results, obtained on well-known and publicly available benchmarks, demonstrate that TG-Pro achieves major performance improvements over other well-established SAT-based ATPG models.

Keywords: *ATPG, Boolean SAT, EDA*

Submitted June 2011; revised November 2011; published January 2012

1. Introduction

TG-Pro is a system for Automatic Test Pattern Generation (ATPG) based on Boolean Satisfiability (SAT). The system integrates the SAT-based core model TG-Pro [3] (TG-Pro 1.0) — *SAT-based **T**est pattern **G**eneration with efficient fault **Pro**pagation constraints*. In contrast to earlier work on SAT-based ATPG [5,9,10,12,14,16], TG-Pro uses a fundamentally different model. With the objective of achieving increased performance, TG-Pro 2.0 develops several optimizations to the core model [3], by exploiting the different aspects of how SAT is used for ATPG.

ATPG has been the subject of extensive research for more than four decades. Initial work focused on structural algorithms [7, 13], and more recently, on SAT. Concrete SAT-based models include NEMESIS [9, 10], TEGUS [16], TG-GRASP [12] and most recently PASSAT [5, 14].

Motivated by the need for more efficient ATPG algorithms for handling large industrial designs, there has been a renewed interest in SAT-based ATPG models and algorithms [4, 5, 14]. Nevertheless, the core SAT-based ATPG model has remained essentially unchanged since the seminal work of T. Larrabee in 1989 [9]. As shown in [3], existing SAT-based ATPG models encode the Boolean difference between the good and the faulty circuits, but add additional variables and constraints for tackling key practical performance drawbacks of the basic Boolean difference model. This paper describes TG-Pro, an ATPG system

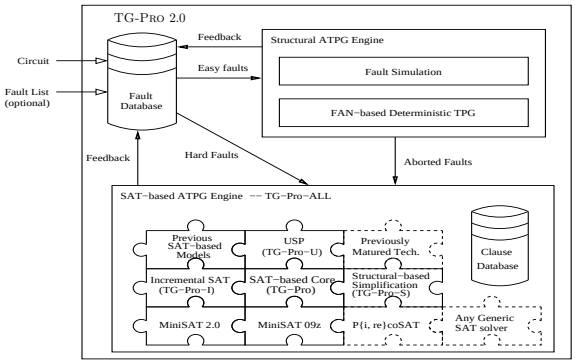

Figure 1. The architecture of TG-Pro 2.0

based on a fundamentally different SAT-based model [3]. Instead of using the standard SAT-based model with *good*, *faulty* and *sensitization* variables, the new model [3] eliminates the *faulty* variables altogether by using a modified semantics for the *sensitization* variables as well as a new formalization of the propagation constraints. The result is a fairly different SAT-based ATPG model, with a significantly smaller number of used variables, and with a negligible increase in the number of used clauses. Experimental results obtained on publicly available industrial benchmarks for ATPG demonstrate that, for individual target faults, TG-Pro can outperform existing ATPG systems by a few orders of magnitude.

2. ATPG

Fabricated Integrated Circuits (ICs) may be subject to defects that may cause circuit failure. Automatic Test Pattern Generation (ATPG) consists of computing input assignments that allow demonstrating the existence or absence of each target fault, or proving no such an assignment exists. An output of ATPG is the classification of the fault: when such an assignment exists, the fault is said to be *detectable*; when no such an assignment exists, the fault is said to be *undetectable*. Undetectable faults are often referred to as *redundant* since undetectable faults are the result of redundancy in circuits [1].

TG-Pro assumes the Single Stuck-at Fault (SSF) model, which is the most widely used model for representing fabrication defects [1]. In the SSF model, a single connection in the circuit is assumed to be stuck at a given logic value, either 0 or 1, denoted respectively by stuck-at 0 (or simply sa-0) and stuck-at 1 (or simply sa-1). Traditional ATPG algorithms [7, 8, 13] exploit the circuit structure. However, it is generally accepted that these algorithms can be ineffective on large industrial circuits [3, 5, 14]. SAT-based ATPG [3, 5, 9, 10, 12, 14, 16] is a well-known alternative, consisting of encoding the ATPG problem into a SAT formula, which can then be solved with a SAT solver.

3. TG-Pro ATPG System 2.0

3.1 System Overview

Figure 1 presents TG-Pro 2.0 architecture. TG-Pro is written in C++ and is compiled into a single binary that simplifies the deployment of the tool. TG-Pro consists of three functional components: *Fault Database*, *SAT-based ATPG Engine* and *Structural ATPG*

Engine. *Fault Database* interacts with the two engines in the way of selecting appropriate faults to suitable engines and analyzing the solving results from the engines. The architecture of the TG-Pro is conventional to the industrial settings, e.g. [5], where faults can be fed to different engines. The *Structural ATPG Engine* applies FAN-based Deterministic TPG [1, 7] and Fault Simulation [1, 11] to tackle the easy faults. The *SAT-based ATPG Engine* is the core component of the system, in which the *hard* faults are solved. The *SAT-based ATPG Engine* of TG-Pro 2.0 is characterized by the following features:

- The new SAT-based model TG-Pro [3] is the kernel of TG-Pro.
- TG-Pro re-implements existing SAT-based ATPG fault propagation models, including NEMESIS [9, 10], TEGUS [16], TG-GRASP [12] and PASSAT [5, 14]. The re-implementation involves key features of their corresponding algorithms but not all.
- TG-Pro implements several additional optimizations:
  TG-Pro-Z: problem-sensitive restarts [15];
  TG-Pro-S: structure-aware simplification;
  TG-Pro-U: static Unique Sensitization Points (USPs) [1, 7];
  TG-Pro-I: incremental CNF formula generation [6];
  TG-Pro-ALL: with all above optimizations.
- TG-Pro interfaces a modern SAT solvers [6, 15] through their APIs. This scheme facilitates exploiting further advances, e.g. [2], in SAT solvers.

3.2 SAT-based ATPG Models

Traditional SAT-based models [5, 9, 10, 12, 14, 16] define *three* variables for each node that is either in the transitive fanout of fault site, or is in the immediate fanin of one of those nodes:

- x^G: Represents the value of the node in the good circuit.
- x^F: Represents the value of the node in the faulty circuit.
- x^S: Encodes the sensitization status.

These three types of variables are used to define constraints for propagating the fault to a primary output. A *good* circuit denotes a circuit without fault and a *faulty* circuit denotes a circuit with fault. x^s is referred to as the *sensitization* variable of node x and takes value 1 only if the value of x differs in the good and the faulty circuits. As indicated before, TG-Pro re-implements standard 3-Variable models [5, 9, 10, 12, 14, 16] and implements the new 2-Variable model [3]. The remainder of this section describes these models.

Standard 3-Variable Model

This category corresponds to the model used in NEMESIS [9, 10], TEGUS [16], TG-GRASP [12] and PASSAT [5, 14]. Given a gate in the fanout cone of a fault, with inputs u and v, output x and fanout nodes y and w, different models encode slightly variant semantics of sensitization variable.

TEGUS Model: In the SAT-based ATPG model used in NEMESIS, TEGUS and PASSAT, the sensitization variable of each node in the fanout cone of the fault is defined as $[x^S \rightarrow (x^G \neq x^F)] \wedge [x^S \rightarrow (y^S \vee w^S)]$. *TG-GRASP Model*: In contrast, in the TG-GRASP model, the sensitization variable is defined as $[x^S \rightarrow (x^G \neq x^F)] \wedge [(x^G \neq x^F) \rightarrow x^S]$.

It is straightforward to conclude that the sensitization variables in the two models have different semantics. Whereas in the NEMESIS, TEGUS and PASSAT model, setting the

sensitization variable to 1 implies that the fault *must* then propagate through that node to a primary output, in the TG-GRASP model the sensitization variable of a node x only indicates whether the fault effect propagates to node x. The different semantics of the sensitization variables leads in practice to somewhat different performance.

New 2-Variable Model

In contrast, TG-Pro [3] uses only two types of variables for each node x in the fanout cone of the fault site. As before, x^G denotes the value in the good circuit. In addition, the new model uses *only* the sensitization variables x^S and avoids the use of the faulty variables x^F. The semantics of x^S is similar to the one used in the TG-GRASP [12] model. This means that setting the sensitization variable of node x to 1 does not necessarily imply propagation to a primary output. Furthermore, given that the faulty variables no longer exist, alternative constraints need to be specified, which do not involve faulty variables. These modified constraints are represented by CNF formulas φ_1 to φ_5 [3]:

- φ_1: If any input of a gate assumes a controlling value [1] and is not sensitized, then the gate output is not sensitized, i.e., $x^S = 0$.
- φ_2: If all inputs of a gate are not sensitized, then the gate output is not sensitized.
- φ_3: If any two inputs are sensitized but their logic values differ, then the output is not sensitized.
- φ_4: For any two inputs of a gate, if one is not sensitized with a non-controlling value [1] while the other one is sensitized, then the output is sensitized.
- φ_5: For any two inputs of a gate, if both are sensitized and their logic value is the same, then the output is sensitized.

These conditions capture all possible conditions for either propagating the fault effect from a gate inputs to the gate outputs, or for blocking propagation. It is straightforward to show that one of the primary outputs becomes sensitized for an input assignment iff the fault is detectable for that assignment.

3.3 System History and Availability

Currently, TG-Pro is the *only* publicly available SAT-based ATPG system being maintained and developed [1]. The first release Ver. 1.0 [3] of TG-Pro fully implements all existing SAT-based ATPG fault propagation models and partially implements their corresponding ATPG algorithms. The second release Ver. 2.0 extends TG-Pro with structural analysis [1, 7, 11], and a number of SAT-based techniques outlined in Section 3.1. The official website of TG-Pro is: http://logos.ucd.ie/wiki/doku.php?id=tg-pro, where the tool, documentation, benchmark files and demos are publicly available.

4. Experimental Results

TG-Pro is implemented in C++. The experimental results of TG-Pro were obtained on a Linux server with 2.93-GHz Intel Xeon processor X3470 and 8-GB RAM. Experiments were performed on publicly available benchmarks for ATPG, namely ISCAS'85, ISCAS'89

1. ATPG in SIS – http://embedded.eecs.berkeley.edu/pubs/downloads/sis/index.htm is publicly available but it outdated and it is no longer maintained.

Table 1. Performance comparison between the representative ATPG systems

Bench.	Circuit Name	#Faults	Total run time (seconds)			
			SPIRIT [8]	TIGUAN [4]	PASSAT [3] [5, 14]	TG-Pro
ISCAS'85	c1908	1879	0.23	0.95	0.64 [14]	0.01
	c2670	2747	0.42	2.60	0.91 [14]	0.11
	c3540	3428	0.20	5.17	3.83 [14]	0.02
	c5315	5350	0.20	3.65	1.44 [14]	0.01
	c6288	7744	0.36	7.61	6.57 [14]	0.01
	c7552	7550	0.58	5.83	3.31 [14]	0.53
ISCAS'89	s9234	6927	1.05	6.47	3.53 [14]	0.61
	s13207	9815	2.78	6.99	3.64 [14]	0.21
	s15850	11725	4.13	12.68	9.18 [14]	0.93
	s35932	39094	5.25	17.28	2.96 [14]	2.95
	s38417	31180	10.98	23.16	4.21 [14]	2.28
	s38584	36303	14.75	22.23	4.84 [14]	1.76
ITC'99 [2]	b14	22802	7.34	23.10	19.00 [5]	2.68
	b15	21988	52.03	66.82	24.00 [5]	27.00
	b17	76625	262.30	252.40	142.00 [5]	248.79
	b18	264047	1555.73	706.92	1350.00 [5]	920.01
	b20	45459	24.52	70.60	56.00 [5]	9.93
	b21	46154	39.67	73.46	59.00 [5]	8.01
	b22	67536	156.00	90.97	95.00 [5]	18.56
Exp. Conf.	SPIRIT	CPU: Intel Xeon X3470 2.93-GHz			MEM: 8-GB	OS: Windows
	TIGUAN	CPU: Intel Xeon X3470 2.93-GHz			MEM: 8-GB	OS: Linux
	PASSAT for ISCAS [3] [14]	CPU: AMD XP 2200+			MEM: 512-MB	
	PASSAT for ITC [3] [5]	CPU: Intel Xeon 3-GHz			MEM: 32-GB	OS: Linux
	TG-Pro	CPU: Intel Xeon X3470 2.93-GHz			MEM: 8-GB	OS: Linux

and ITC'99 [2] (see [3] and TG-Pro official website for details). For each circuit, *all* faults were targeted. Due to space restrictions, only representative circuits are shown.

Table 1 compares the performance of the representative ATPG systems, namely the structural ATPG system SPIRIT [8], the thread-parallel ATPG system TIGUAN [4], the most recent SAT-based ATPG systems PASSAT [5,14], and TG-Pro. In the experiments, TG-Pro accomplished the testing of all ISCAS'85, ISCAS'89 and ITC'99 [2] benchmarks with *zero* aborted faults. As can be concluded, supported by experimental data obtained on well-known and publicly available industrial benchmarks for ATPG, TG-Pro achieves observable performance improvements over the other well-established ATPG systems.

5. Conclusion

SAT-based ATPG motivated in part the development of modern SAT solvers. Nevertheless, the core SAT-based model for ATPG has remained unchanged for almost two decades, even in the presence of the renewed interest in applying SAT to industrial circuits [5, 14]. The TG-Pro ATPG system integrates a recent and new SAT-based ATPG model [3], and develops several new techniques for the optimizing the use of SAT solvers. Experimental results demonstrate that TG-Pro outperforms other well-established ATPG systems.

Acknowledgement This work is partially supported by European project COCONUT (FP7-ICT-217069), and SFI PI grant BEACON (09/IN.1/I2618).

References

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

2. The second release is available from http://www.cad.polito.it/tools/itc99.html. Observe that the number of collapsed faults is *larger* than in the first release.
3. Unfortunately, PASSAT [5, 14] was unavailable from the authors. The experimental results shown are quoted from the corresponding papers, where ISCAS'85 & ISCAS'89 results are quoted from [14] and ITC'99 results are quoted from [5].

[2] Armin Biere. PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, **4**:75–97, 2008.

[3] Huan Chen and Joao Marques-Silva. TG-Pro: A new model for SAT-based ATPG. In *IEEE International High Level Design Validation and Test Workshop*, pages 76–81, 2009.

[4] 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 *International conference on VLSI Design*, pages 227–232, 2009.

[5] R. Drechsler, S. Eggersglu, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille. On acceleration of SAT-based ATPG for industrial designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **27**(7):1329–1333, 2008.

[6] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, pages 502–518, 2003.

[7] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, **32**(12):1137–1144, 1983.

[8] E. Gizdarski and H. Fujiwara. SPIRIT: a highly robust combinational test generation algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **21**(12):1446–1458, 2002.

[9] T. Larrabee. Efficient generation of test patterns using Boolean difference. In *International Test Conference*, pages 795–801, 1989.

[10] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **11**(1):4–15, 1992.

[11] H. K. Lee and D. S. Ha. Atalanta: An efficient ATPG for combinational circuits. Technical report, 93-12, DEE, Virginia Polytechnic Insitute and State Univ., 1993.

[12] J. Marques-Silva and K.A. Sakallah. Robust search algorithms for test pattern generation. In *International Symposium on Fault-Tolerant Computing*, pages 152–161, 1997.

[13] J. P. Roth. Diagnosis of automata failures: a calculus and a method. *IBM Journal of Research and Development*, **10**:278–291, 1966.

[14] Junhao Shi, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schloffel. PASSAT: efficient SAT-based test pattern generation for industrial circuits. In *IEEE annual symposium on VLSI*, pages 212–217, 2005.

[15] Carsten Sinz and Markus Iser. Problem-sensitive restart heuristics for the DPLL procedure. In *Theory and Applications of Satisfiability Testing*, pages 356–362, 2009.

[16] P. Stephan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **15**(9):1167–1176, 1996.