

ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ И ИХ ЭЛЕМЕНТЫ/COMPUTING SYSTEMS AND THEIR ELEMENTS

DOI: https://doi.org/10.60797/itech.2026.11.8 EDN: BEAXLC

SAT-BASED STUCK-AT ATPG ON ELABORATED RTL WITHOUT LOGIC SYNTHESIS: AN EXPERIMENTAL VALIDATION STUDY AND OUTLOOK FOR EARLY-STAGE BLOCK CHARACTERIZATION

Research article

Muzalevsky Y.Y.[1, *]

[1] National Research University of Electronic Technology, Moscow, Russian Federation

* Corresponding author (muzalevskij-8g[at]yandex.ru)

Suggested: 08.05.2026; Accepted: 13.07.2026; Published: 14.07.2026

Abstract

A SAT-based automatic test pattern generation (ATPG) flow operating directly on the And-Inverter Graph (AIG) extracted from the elaborate stage of register-transfer-level (RTL) description — without invoking logic synthesis — is investigated. Test vectors generated by a SAT solver on the elaborated representation are conjectured to detect the same stuck-at faults in the synthesized gate-level netlist whenever the two representations are formally proven equivalent by logical-equivalence checking (LEC). The conjecture is examined experimentally on six combinational and sequential benchmarks of growing complexity using a fully open-source toolchain (Yosys, MiniSAT). For all benchmarks, both flows reach 100% fault coverage of the testable subset; logical equivalence is proven by Yosys for every case. A cross-validation protocol — applying vectors obtained from the synthesis-free flow to the synthesized AIG and vice versa — confirms complete transferability of test vectors for all combinational benchmarks. The redundant-logic benchmark exposes a discriminating effect: the synthesis-free flow detects two redundant stuck-at faults that synthesis collapses, providing a redundancy signal at the elaborate stage that is otherwise lost. The validated infrastructure opens a direct path to early-stage block characterization: properties traditionally measured after synthesis (testable fault count, vector-set length, redundancy indicators) become available at the RTL elaborate stage, where they can inform design decisions before synthesis is invoked.

Keywords: SAT, ATPG, RTL, stuck-at, elaborate, AIG, LEC, Yosys, MiniSAT, design for testability, early-stage analysis.

SAT-ГЕНЕРАЦИЯ ТЕСТОВЫХ ВЕКТОРОВ НА ELABORATE-ПРЕДСТАВЛЕНИИ RTL БЕЗ ЭТАПА ЛОГИЧЕСКОГО СИНТЕЗА: ЭКСПЕРИМЕНТАЛЬНОЕ ОБОСНОВАНИЕ МАРШРУТА И ПЕРСПЕКТИВЫ РАННЕЙ ХАРАКТЕРИЗАЦИИ БЛОКОВ

Научная статья

Музалевский Я.Ю.[1, *]

[1] Национальный исследовательский университет «Московский институт электронной техники», Москва, Российская Федерация

* Корреспондирующий автор (muzalevskij-8g[at]yandex.ru)

Предложена: 08.05.2026; Принята: 13.07.2026; Опубликована: 14.07.2026

Аннотация

Исследуется маршрут SAT-генерации тестовых векторов (ATPG) на основе классической модели stuck-at, применяемый непосредственно к And-Inverter Graph (AIG), полученному из этапа elaborate RTL-описания, без выполнения логического синтеза. Выдвигается гипотеза: тестовые векторы, найденные SAT-решателем на elaborate-AIG, обнаруживают те же stuck-at неисправности в синтезированном вентильном нетлисте при условии формально доказанной логической эквивалентности (LEC) обоих представлений. Гипотеза проверена экспериментально на шести комбинаторных и последовательных бенчмарках возрастающей сложности с использованием полностью открытого инструментального маршрута (Yosys, MiniSAT). Для всех бенчмарков оба маршрута достигают 100% покрытия тестируемых неисправностей, а LEC-эквивалентность доказана средствами Yosys. Перекрёстная валидация — применение векторов из синтез-свободного маршрута к синтезированному AIG и наоборот — подтвердила полную переносимость тестовых векторов для всех комбинаторных схем. Бенчмарк с избыточной логикой выявил различающий эффект: синтез-свободный маршрут обнаруживает две избыточные неисправности, которые синтез скрывает, давая инженеру сигнал о тестируемости на этапе elaborate. Подтверждённый маршрут открывает прямой путь к ранней характеризации блоков: свойства, традиционно измеряемые после синтеза (число тестируемых неисправностей, длина набора векторов, индикаторы избыточности), становятся доступны на этапе elaborate и могут влиять на проектные решения до запуска синтеза.

Ключевые слова: SAT, ATPG, RTL, stuck-at, elaborate, AIG, LEC, Yosys, MiniSAT, проектирование с учётом тестируемости, ранний анализ тестируемости.

Introduction

Design for testability (DFT) is integral to integrated-circuit design. Among DFT tasks, automatic test-pattern generation (ATPG) for stuck-at faults is one of the most computationally demanding stages of the verification flow [1], [2]. Conventionally, ATPG operates on a synthesized gate-level netlist: the RTL description is first compiled by a logic-synthesis

tool, then technology-mapped to a standard-cell library, and only afterwards passed to the ATPG engine. The synthesis step itself is computationally expensive — on industrial system-on-chip designs it consumes minutes to hours per iteration.

For a DFT engineer iterating on the RTL to improve testability — adding test points, restructuring scan chains, modifying control logic — the dependence on a full synthesis cycle creates a long feedback loop: a single RTL modification cannot be evaluated for testability without re-running synthesis. The cumulative cost of these iterations dominates the early stages of DFT exploration.

The present work investigates whether the ATPG problem can be formulated and solved directly on the Boolean representation produced by the elaborate stage of RTL, prior to logic synthesis. The hypothesis is that test vectors found by a SAT solver on the AIG of elaborated RTL are sufficient to detect stuck-at faults in the synthesized netlist, provided the two representations are proven logically equivalent.

Existing work on RTL-level testability addresses a related but distinct question. P.A. Thaker, V.D. Agrawal and M.E. Zaghloul [3] propose a register-level testability methodology; Corno, Sonza Reorda and Squillero [6] introduce the ITC'99 RTL benchmarks. Strauch's GIF model [4], [5] derives faults from the internal structure of complex RTL operators and demonstrates that 100% GIF coverage on RTL implies 100% stuck-at coverage at the gate level — a result that conceptually parallels the present work but adopts a different fault model. SAT-based RTL satisfiability has been studied in LPSAT [7] and USAT [8]. Larrabee's original SAT-ATPG construction [1], extended in [2], [9], [10], [11], remains the canonical formulation. Despite this body of work, no published study uses formal LEC as the bridge for transferring SAT-ATPG results obtained on the elaborated AIG of RTL to the synthesized gate-level netlist with experimental validation.

The contributions of this paper are: (i) a methodological flow combining Yosys and MiniSAT in a Larrabee-style SAT-ATPG pipeline operating on elaborated RTL; (ii) an experimental cross-validation protocol that quantifies vector transferability between the two representations independently of LEC; (iii) an outline of early-stage block-characterization use cases enabled by the validated flow.

Research methods and principles

Tool flow

Two parallel flows are considered (Fig. 1). Route A (reference) processes the RTL through full Yosys synthesis (synth -flatten), maps the netlist to AIG via abc -g AND, and feeds the result into the SAT-ATPG engine. Route B (proposed) processes the RTL through proc; flatten; opt only, then maps the result to AIG via the same abc -g AND step, and feeds it into the same SAT-ATPG engine. Sequential designs use the clk2fflogic pass to encode flip-flops as AIGER state elements. The full tool stack is open-source: Yosys 0.45+42, MiniSAT 2.2.0, Python 3.12; experiments run on x86_64 Ubuntu 24.04, Intel Core i5, 16 GB RAM. A 30-second per-fault timeout is imposed on the SAT solver. Test-vector compaction is not applied: one vector per fault.

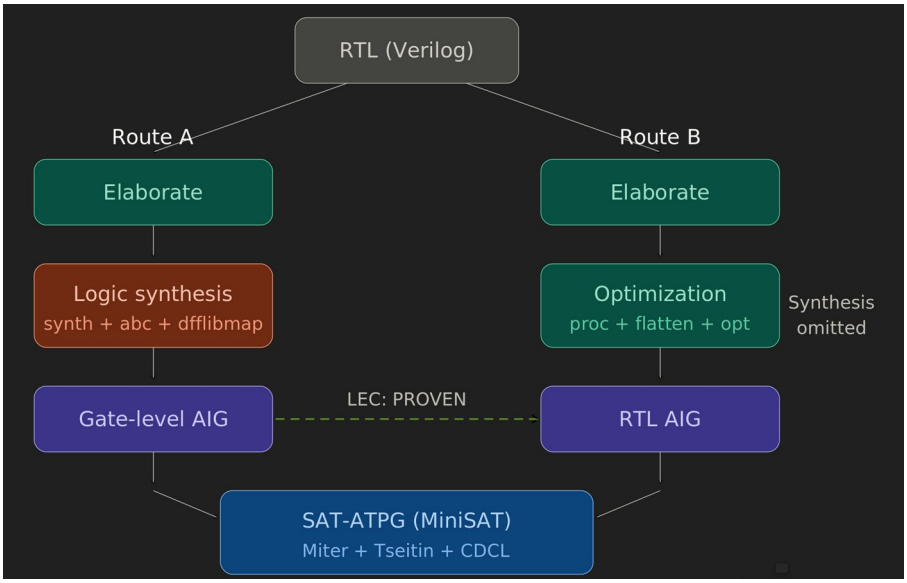

Figure 1 - Tool flow: Route A (full synthesis) versus Route B (elaborate only)
DOI: https://doi.org/10.60797/itech.2026.11.8.1

Fault model and miter construction

The classical stuck-at fault model is applied to every variable in the AIG: primary inputs, AND-node outputs, primary outputs. For each conductor, both stuck-at-0 and stuck-at-1 faults are considered. The Larrabee miter is constructed in conjunctive normal form (CNF) for each fault (Fig. 2): two copies of the circuit (good and faulty) are encoded by Tseitin transformation; in the faulty copy the driving AND clauses for the faulted conductor are cut (omitted) and the conductor is forced to the stuck value; corresponding primary inputs of the two copies are joined by equality clauses (skipped for the faulted primary input itself); outputs are compared by an XOR network with at least one mismatch demanded. The CNF is solved by MiniSAT. SAT returns a vector that activates and propagates the fault; UNSAT certifies the fault as untestable (redundant).

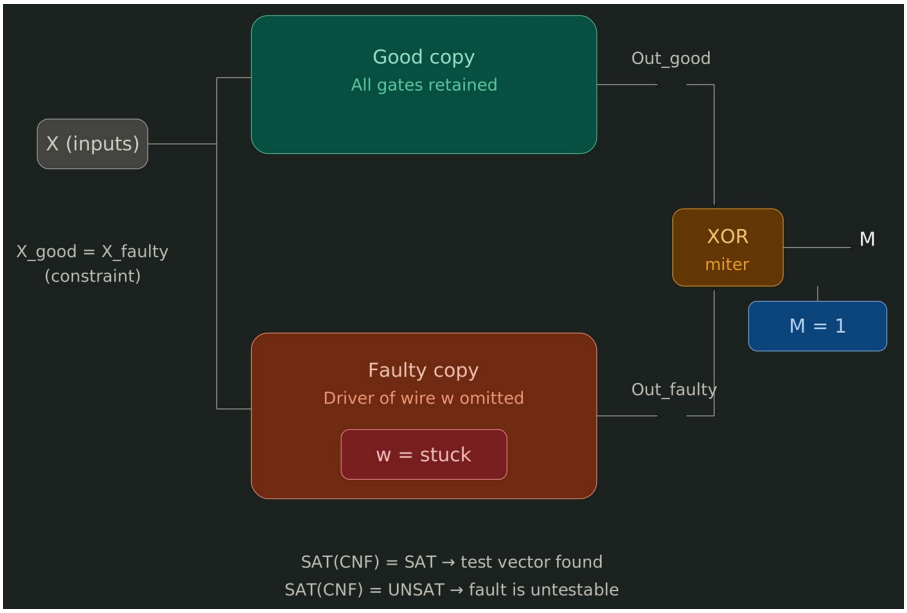

Figure 2 - Larrabee miter construction for stuck-at fault detection
DOI: https://doi.org/10.60797/itech.2026.11.8.2

LEC and cross-validation

Logical equivalence between Route A and Route B is established by Yosys via equiv_make, equiv_simple, equiv_induct and equiv_status. Equivalence implies transferability of test vectors for functional faults (stuck-at on primary inputs and primary outputs): for any input vector the two functions agree, hence a vector that activates a functional fault in one representation activates the corresponding fault in the other. For internal faults — stuck-at on intermediate AND-node outputs — transferability does not follow from LEC because the fault universes differ structurally: the two AIGs realize the same Boolean function with different internal wiring. The cross-validation protocol fills this gap empirically: each Route B test vector is applied to Route A's AIG by topological simulation against a faulted copy, and the fraction of Route A's testable faults thereby detected is recorded; the dual measurement is performed in the opposite direction. A separate restricted measurement is reported for the PI/PO subset, where LEC theoretically guarantees 100% transferability.

Benchmarks

Six benchmarks of increasing complexity are used. bench_xor_a and bench_xor_b are two textually different yet semantically identical implementations of the XOR function (operator form (a|b)&~(a&b) versus the ^ operator) — a sanity probe for invariance of the SAT result under equivalent RTL spellings. bench_alu is a 16-bit arithmetic-logic unit; bench_shifter is a 22-bit barrel shifter; bench_redundant is a majority-of-three function with one logically redundant product term; bench_counter is a synchronous 8-bit counter with comparator — the only sequential benchmark in the set.

Main results

Fault coverage

Table 1 reports stuck-at fault coverage for both flows. Fault coverage is defined as FC = det / (total − unt) × 100%, where det is the number of detected faults, total is the size of the fault universe, and unt is the number of provably untestable (redundant) faults. All six benchmarks reach 100% coverage of the testable subset on both flows.

Table 1 - Stuck-at fault coverage

DOI: https://doi.org/10.60797/itech.2026.11.8.3

Benchmark	A: det/total	A: FC, %	A: unt	B: det/total	B: FC, %	B: unt
bench_xor_a	10/10	100.00	0	10/10	100.00	0
bench_xor_b	10/10	100.00	0	10/10	100.00	0
bench_alu	480/480	100.00	0	474/474	100.00	0
bench_shifter	1110/1112	100.00	2	1142/1142	100.00	0
bench_redundant	16/16	100.00	0	20/22	100.00	2
bench_counter	200/200	100.00	0	200/200	100.00	0

bench_xor_a and bench_xor_b, despite their different RTL spellings, produce byte-identical AIGER files after the elaborate stage (verified by SHA-256 hash equality), confirming the textual-form invariance of the SAT formulation. The differences in the total number of faults between flows on bench_alu, bench_shifter and bench_redundant reflect the different

number of internal AND nodes; the testable coverage is identical. bench_redundant is the only benchmark where the unt counts diverge in a non-trivial direction: Route B reports two untestable faults — the stuck-at faults on the redundant majority term t4 = a & b & c, masked by the remaining terms; Route A reports zero, because synthesis collapses the redundant logic before ATPG sees it.

Timing

Table 2 reports per-benchmark wall-clock time. The Yosys preparation time of Route B is 20–65% of Route A's. On the present small benchmarks the SAT-solver time dominates the end-to-end flow, and the absolute speed-up is modest. On industrial designs where synthesis takes hours, the reduction in preparation time is expected to translate into a substantially shorter feedback loop.

Table 2 - Wall-clock time

DOI: https://doi.org/10.60797/itech.2026.11.8.4

Benchmark	A: Yosys, s	A: ATPG, s	A: total, s	B: Yosys, s	B: ATPG, s	B: total, s	**A/B**
bench_xor_a	0.074	0.040	0.114	0.048	0.050	0.098	1.16×
bench_xor_b	0.068	0.030	0.098	0.040	0.030	0.070	1.40×
bench_alu	0.137	3.010	3.147	0.090	3.200	3.290	0.96×
bench_shifter	0.232	11.760	11.992	0.144	12.610	12.754	0.94×
bench_redundant	0.080	0.050	0.130	0.016	0.110	0.126	1.03×
bench_counter	0.113	0.750	0.863	0.075	0.780	0.855	1.01×

Cross-validation

Table 3 reports cross-validation results. For all five combinational benchmarks, vectors obtained on Route B detect 100% of testable faults of Route A and vice versa. The single deviation is bench_redundant in the A→B direction (90.91%): vectors from the synthesized flow cannot detect Route B's two redundant faults, which by definition have no input vector activating them. Restricted to the PI/PO subset, transferability is 100% in all directions, in agreement with the LEC-backed theoretical claim. bench_counter is excluded because the latch encoding of state in AIGER is not preserved under the two flows; aligning latch correspondences across representations is left as future work.

Table 3 - Cross-validation: detection of one flow's faults by the other flow's vectors

DOI: https://doi.org/10.60797/itech.2026.11.8.5

Benchmark	Vec B	Vec A	FC B→A, %	B→A PI/PO, %	FC A→B, %	A→B PI/PO, %
bench_xor_a	10	10	100	100	100	100
bench_xor_b	10	10	100	100	100	100
bench_alu	474	480	100	100	100	100
bench_shifter	1142	1110	100	100	100	100
bench_redundant	20	16	100	100	90.91	100
bench_counter	—	—	—	—	—	—

AIG structure

Table 4 reports AIG-level metrics. The two flows produce structurally distinct AIGs of comparable size; for bench_counter the two AIGs are byte-identical because the increment-and-compare logic is already in canonical form and abc introduces no structural change. bench_redundant exhibits the largest structural divergence (5 vs 8 AND nodes), reflecting the redundancy collapsed by synthesis.

Table 4 - AIG structure and CNF size

DOI: https://doi.org/10.60797/itech.2026.11.8.6

Benchmark	A: I/O/R/AND	A: depth	B: I/O/R/AND	B: depth	A: CNF v/c	B: CNF v/c
bench_xor_a	2/1/0/3	2	2/1/0/3	2	6/9	6/9
bench_xor_b	2/1/0/3	2	2/1/0/3	2	6/9	6/9
bench_alu	19/9/0/221	19	19/9/0/218	20	241/663	238/654
bench_shifter	22/21/0/534	12	22/21/0/549	12	557/1602	572/1647
bench_redundant	3/1/0/5	3	3/1/0/8	4	9/15	12/24
bench_counter	11/10/18/89	12	11/10/18/89	12	119/267	119/267

Legend: I — primary inputs; O — primary outputs; R — registers (D-flip-flops, encoded as AIGER state elements via clk2fflogic); AND — AIG AND-node count; depth — longest path from primary inputs to primary outputs; v/c — CNF variable / clause counts.

LEC

For all six benchmarks equiv_status -assert returns PROVEN. The combinational benches use a single equiv_simple invocation; bench_counter requires async2sync followed by equiv_induct -seq 10 for the sequential proof.

Discussion

The experimental evidence supports the central hypothesis: SAT-ATPG operating on the AIG produced by elaborated RTL — without logic synthesis — yields test vectors that detect stuck-at faults in the synthesized netlist with no measurable degradation in fault coverage of the testable subset. Logical equivalence is formally proven; cross-validation independently confirms vector transferability. The position relative to Strauch's GIF model [4], [5] is conservative: GIF proposes a new RTL-level fault model derived from the structure of complex RTL operators; the present method retains the standard stuck-at model on AIG and uses formal LEC as the bridge. This makes the method directly compatible with industrial coverage metrics and with existing gate-level ATPG pipelines that consume stuck-at coverage reports.

The validated infrastructure shifts where DFT evaluation lives within the design loop — from a post-synthesis verification step to an early RTL-exploration step. Three classes of use cases follow directly. They are outlined here as enabled possibilities; concrete optimization algorithms for any of them are left as separate future contributions.

(i) Test-count reduction by RTL-stage redundancy detection. The bench_redundant result shows that elaborate-stage SAT-ATPG identifies untestable redundant logic that synthesis would silently collapse. Surfacing this signal early lets the engineer either accept the redundancy intentionally (for hazard immunity, glitch suppression) or rewrite the RTL to expose the optimization explicitly to a downstream tool flow.

(ii) Block-level effort estimation for built-in self-test (BIST) planning. The cardinality of the resulting test-vector set is a direct effort metric: which RTL block needs more BIST bandwidth, which fits within a tight pattern budget. Today this metric is available only after synthesis; the present flow makes it available at the elaborate stage.

(iii) Comparative pre-synthesis evaluation of competing RTL implementations. The bench_xor_a versus bench_xor_b probe demonstrates that semantically identical RTL written in different operator forms produces byte-identical AIGER output, hence identical SAT-ATPG outcomes. Applied to non-trivially different RTL implementations of the same function, the same probe directly reveals which variant has the smaller AIG, smaller fault universe, or fewer redundant nodes — properties that until now could only be assessed after synthesis.

The common pattern across (i)–(iii) is a re-positioning of testability evaluation within the design flow: tasks traditionally performed at the end (after synthesis) become accessible at the start (after elaborate). The expectation, supported by the present results, is that performing these evaluations early — when an RTL change can still be made cheaply — yields substantively different design decisions than performing them late, when the same change demands a full re-synthesis.

Limitations are explicit. First, the LEC-backed transferability claim is rigorous for functional faults (PI/PO) and empirical for structural-internal faults; for designs with substantially different AIG topologies the empirical claim should be re-verified per design. Second, fault models requiring structural information (transition delay, cell-aware, bridging) remain the prerogative of gate-level ATPG. Third, the present sequential treatment via clk2fflogic covers full-scan-equivalent topologies; partial-scan, multi-clock-domain and BIST-instrumented designs require additional handling. Fourth, scalability to industrial-size projects requires confirmation on benchmarks of order 10^5–10^6 gates, where SAT-solver time may dominate even when synthesis cost is removed.

Conclusion

A SAT-based stuck-at ATPG flow operating on elaborated RTL — without logic synthesis — has been experimentally validated on six benchmarks using a fully open-source toolchain. All benchmarks reached 100% fault coverage of the testable subset on both the synthesis-based reference flow and the synthesis-free proposed flow; logical equivalence between the two representations was formally proven by Yosys for every case; cross-validation confirmed full transferability of test vectors between the two representations on all combinational benchmarks. A discriminating exception on the redundant-logic benchmark demonstrated that the synthesis-free flow exposes redundancy signals that are otherwise collapsed by synthesis.

The validated infrastructure shifts testability evaluation from a post-synthesis verification step to an early RTL-exploration step, and outlines a direct path to early-stage block characterization use cases — test-count reduction, BIST effort estimation, and comparative RTL-variant evaluation — that until now required a full synthesis cycle to access. Concrete optimization algorithms enabled by this re-positioning are the subject of subsequent work in the series.

Конфликт интересов

Не указан.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Conflict of Interest

None declared.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы на английском языке / References in English

1. Larrabee T. Test pattern generation using Boolean satisfiability / T. Larrabee // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. — 1992. — Vol. 11. — № 1. — P. 4–15. — DOI: 10.1109/43.108614.

2. Kunz W. Sat and ATPG: Algorithms for Boolean Decision Problems / W. Kunz, J.P. Marques-Silva, S. Malik // Logic Synthesis and Verification. — Boston : Springer, 2002. — P. 309–341. — DOI: 10.1007/978-1-4615-0817-5_12.

3. Thaker P.A. Register-transfer level fault modeling and test evaluation techniques for VLSI circuits / P.A. Thaker, V.D. Agrawal, M.E. Zaghloul // Proceedings of International Test Conference (ITC). — Atlantic City : IEEE, 2000. — P. 940–949. — DOI: 10.1109/TEST.2000.894305.

4. Strauch T. A novel RTL ATPG model based on Gate Inherent Faults (GIF-PO) of complex gates / T. Strauch // arXiv. — 2016. — URL: https://arxiv.org/abs/1612.05166 (accessed: 01.05.2026).

5. Strauch T. An RTL ATPG flow using the Gate Inherent Fault (GIF) model applied on non-, standard- and random-access-scan (RAS) / T. Strauch // Proceedings of IEEE International Test Conference (ITC). — Washington : IEEE, 2019. — P. 1–10. — DOI: 10.1109/DSD.2019.00018.

6. Corno F. RT-level ITC'99 benchmarks and first ATPG results / F. Corno, M.S. Reorda, G. Squillero // IEEE Design & Test of Computers. — 2000. — Vol. 17. — № 3. — P. 44–53. — DOI: 10.1109/54.867894.

7. Zeng Z. LPSAT: A unified approach to RTL satisfiability / Z. Zeng, P. Kalla, M. Ciesielski // Proceedings of Design, Automation and Test in Europe (DATE). — Munich : IEEE, 2001. — P. 186–190. — DOI: 10.1109/DATE.2001.915055.

8. Wu W.-M. USAT: An integrated platform for satisfiability solving and model checking / W.-M. Wu, M.-C. Chen // Proceedings of IEEE International Conference on Computer Design (ICCD). — Lake Tahoe : IEEE, 2008. — P. 440–445. — DOI: 10.1109/CSSE.2008.1352.

9. Kunz A. SAT and ATPG: Boolean engines for formal hardware verification / A. Kunz, W. Biere // Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD). — San Jose : IEEE, 2002. — P. 782–785. — DOI: 10.1109/ICCAD.2002.1167620.

10. Drechsler S. Improved SAT-based ATPG: More constraints, better compaction / S. Drechsler, S. Eggersglüß, R. Drechsler // Proceedings of Asian Test Symposium (ATS). — Yilan : IEEE, 2013. — P. 85–90. — DOI: 10.1109/ICCAD.2013.6691102.

11. Tille D. A fast untestability proof for SAT-based ATPG / D. Tille, R. Drechsler // Proceedings of Design, Automation and Test in Europe (DATE). — Nice : IEEE, 2009. — P. 1–6. — DOI: 10.1109/DDECS.2009.5012096.

12. Wolf C. Yosys — A free Verilog synthesis suite / C. Wolf // Proceedings of Austrochip Workshop. — 2013. — P. 97–101. — URL: https://yosyshq.net/yosys/files/yosys-austrochip2013.pdf (accessed: 01.05.2026).

13. Eén N. An extensible SAT-solver / N. Eén, N. Sörensson // Theory and Applications of Satisfiability Testing (SAT). Lecture Notes in Computer Science, vol. 2919. — Berlin ; Heidelberg : Springer, 2003. — P. 502–518. — DOI: 10.1007/978-3-540-24605-3_37.

14. Biere A. The AIGER And-Inverter Graph (AIG) format version 20071012 / A. Biere. — 2007. — URL: https://fmv.jku.at/aiger/FORMAT.aiger (accessed: 01.05.2026).

15. Brayton R. ABC: An academic industrial-strength verification tool / R. Brayton, A. Mishchenko // Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 6174. — Berlin ; Heidelberg : Springer, 2010. — P. 24–40. — DOI: 10.1007/978-3-642-14295-6_5.