INCREMENTAL AUTOMATIC CORRECTION FOR DIGITAL VLSI CIRCUITS

Lamya Gaber[1], Aziza I. Hussein[2] and Mohammed Moness[1]

[1]Department of Computers and Systems Engineering,
Minia University, Minia, Egypt
[2]Department of Electrical and Computer Engineering,
Effat University, Jeddah, KSA

ABSTRACT

The impact of the recent exponential increase in complexity of digital VLSI circuits has heavily affected verification methodologies. Many advances toward verification and debugging techniques of digital VLSI circuits have relied on Computer Aided Design (CAD). Existing techniques are highly dependent on specialized test patterns with specific numbers increased by the rising complexity of VLSI circuits. A second problem arises in the form of large sizes of injecting circuits for correction and large number of SAT solver calls with a negative impact on the resultant running time. Three goals arise: first, diminishing dependence on a given test pattern by incrementally generating compact test patterns corresponding to design errors during the rectification process. Second, to reduce the size of in-circuit mutation circuit for error-fixing process. Finally, distribution of test patterns can be performed in parallel with a positive impact on digital VLSI circuits with large numbers of inputs and outputs. The experimental results illustrate that the proposed incremental correction algorithm can fix design bugs of type gate replacements in several digital VLSI circuits from ISCAS'85 with high speed and full accuracy. The speed of proposed Auto-correction mechanism outperforms the latest existing methods around 4.8x using ISCAS'85 benchmarks. The parallel distribution of test patterns on digital VLSI circuits during generating new compact test patterns achieves speed around 1.2x compared to latest methods.

KEYWORDS

Auto-correction, ATPG, Fault detection, Verification.

1. INTRODUCTION

Recently, integrated circuits (ICs) known as microchips have become omnipresent in all aspects of our lives from automobiles to smartphones. Their impact on our lives has gradually evolved to be the most crucial chips as they are considered the heart of modern computing devices. These compact ICs contain billions of transistors which perform billions of computations compared with vacuum tubes or single transistor preceded them. Therefore, the complexity of ICs design flow has increased over the last three decades with a highly increasing list of requirements in terms of functional, power, performance and physical size. Therefore, the IC design is not a push-button process. Instead, it is a highly complicated task as it needs a full understanding of the IC restrictions, specifications and all the required EDA tools. Several procedures should be considered in the IC design flow which are highly error-prone as many translations should be performed on various levels of IC representations (e.g. RTL description and gate-level netlist). Therefore, the uselessness rate of silicon slabs has been increased by increasing the number of undetected errors that might propagate from the RTL description to the fabricated chip. Hence,

David C. Wyld et al. (Eds): ACITY, DPPR, VLSI, WeST, DSA, CNDC, IoTE, AIAA, NLPTA - 2020
pp. 83-93, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101508

several recent contributions of developing verification and testing methodologies have been emerged to avoid the consumed cost and time for achieving the desired design with its required specifications. And by the time, the procedure of verifying, testing, debugging and correcting digital circuits have been contributing up to 70% of the entire time for designing the aspired IC [1]. On top of that, the functional verification has become the most significant issue of the IC design as the fact proved in [2] that the main reason for unsuccessful Application Specific Integrated Circuits (ASIC), around 60% of them, is the existence of undetected functional errors (Not because of high power consumptions and timing issues). Consequently, up to 46% of the whole time of IC design is consumed in the functional verification step [3]. Therefore, three main processes are implemented following each conversion from high level to more detailed level of abstraction in the pre-silicon design, termed Verification, Debugging and Correction. These processes are defined as follows:

Definition 1: the process of searching for inconsistencies between two levels of circuit abstraction in the pre-silicon design is called "*Verification*".

Definition 2: In case of functional inconsistencies intended for diagnosing and detecting potential bug locations in an erroneous circuit is called "*Debugging*". Therefore, it is also often termed *Bug Localization.*

Definition 3: the phase responsible for modifying components causing errors discovered by debuggers is known as "*Correction*", so it can rectify the desired circuit to behave in its intended manner. The output of this phase is the correct design that can match the desired specification (the previous abstraction).

By increasing the design size, the uselessness rate of silicon slabs has been grown by increasing the number of undetected errors that might propagate from the RTL description to the fabricated chip. In addition, the proportion of losing time and increasing the non-recurring engineering (NRE) cost is gradually extended by taking the blind assuming of "synthesis tools make no mistakes". Therefore, several recent contributions of developing verification/testing, debugging and correction methodologies have been emerged to avoid the cost and time consumed during achieving the desired design that should be matched with the behavioral specifications. And by the time, the procedures of verifying/testing, debugging and correcting digital circuits have been contributing up to 70% of the entire time for designing the aspired IC [1]. As the growth of the complexity of digital integrated circuits and reduced time-to-market budget, debugging mechanisms with auto-correction in a digital design flow have become more and more challenging task, contributing on average 60% of the verification process in case of observing some failures in a given design. Although, many contributions have been devoted on the verification and debugging steps, few efforts have been dedicated on fixing the detected errors which are left to the creativities of the designers to manually correct them. Also, studies report that the main reasons for failures existed in a large portion of first tap-out are simply functional errors which could escape from the earlier correction step [1]. So, the correction procedure is considered the expensive and complicated task because other phases of IC design directly depend on "how efficiently the correction procedure works". In this paper, we focus on improving the process of auto-correction using partial test patterns in the gate-level representation as it is a core factor of reducing the ad-hoc manual effort, time and possibilities of undetected bugs in the manufactured chip.

Gate-level auto-correction problem has been addressed by many researches proposing different algorithms for auto-debugging and/or suggesting the possible modification for achieving the correct design. Also, most of error correction techniques are focused on gate-level netlist because it can give the designers different reasons of occurring bugs such as incorrect individual

connection wires or incorrect individual gate types that can NOT be provided in a higher level of abstraction. The conceivable idea of fixing errors by re-synthesis is not an efficient way for most of today's digital circuits as errors can be caused by the synthesis tool itself. Also, some physical optimization that should be previously performed might Not be validated.

In this paper, we proposed a new efficient automatic correction (ACM-CTV) method using partial test patterns to generate both a corrected digital circuit and compact test patterns by the following advantages:

1- The injecting circuit is reduced to be 3-to-1 multiplexers instead of 6-to-1 multiplexers as it used in [4].
2- One of two solutions that are generated in every iteration is exploited to reduce the search space (*Find_OneSol* Algorithm).
3- Parallel distribution of test patterns (*GPU_UC* procedure) is performed that can give a high performance in case of large number of inputs and outputs of faulty digital circuit.

2. RELATED WORK

Recently, many extensive problems in digital very-large-scale-integration problems can be addressed within a Boolean satisfiability framework as it offers various solutions. In [5-7], the main idea is mapping the diagnosis problem to SAT problem in a compatible form (as CNF) to be solved using SAT solvers. These SAT-based methods outperform the traditional methods as it is implemented with high performance.

In [8], the bug localization problem is addressed by replacing every possibly buggy gate with 2-to-1 multiplexers, then converting to a conjunction normal form (CNF) formula. After that, a new constraint is added to the CNF instance to represent the initial test patterns and passing all the final formula to SAT solver to return a satisfiability model of CNF instance that can detect the exact locations.

Authors in [6] have proposed a debugging method of gate-level circuits using partial maximum satisfiability (MAX-SAT) for detecting spatial and temporal bug locations.

In [9], authors proposed a bug localization of gate-level circuits using level Quantified Boolean Formula (QBF) that is equivalent process to repetitive calls of normal SAT solvers. All the previous methods attempt to exactly detect logic bugs in order to finally rectify the digital circuit to satisfy specifications.

On the other hand, the auto-correction methods have been addressed using both the bug locations and test patterns for the corresponding logic bugs. In [10], a mutation-based correction mechanism is proposed by adding 6-to-1 multiplexers into the place of every potential bug to quickly check all possible gates instead of the faulty one. In [11], the rectification process is employed by injecting 3-to-1 multiplexers instead of 6-1 multiplexers in order to reduce the CNF instance passed to SAT engine for shrinking the search space. The previous two methods are based on knowing exact bug locations and test patterns of the correct digital circuit in order to automatically correct circuits. In [12], a new correction method is proposed based on [10] in order to incrementally correcting a given circuit by generating new test patterns. The generation of compact test patterns is performed by finding two different solutions in every iteration, so it can guarantee that the returned rectified circuit is completely corrected for all test patterns.

In [4], authors improved the previous automatic correction method by exploiting the two solutions generated in every iteration not only in finding a new test pattern but also, in shrinking the search space for the next iterations. Therefore, the proposed rectification (ACM-CTV) mechanism exploit advantages of methods proposed in [4, 11] in order to reducing search space and running time for correction.

3. BACKGROUND

3.1. Design Errors

A component among a design implementation such as RTL or gate-level representation or behavioural specifications such as testbench and assertions that produces functional inconsistencies between the two representation is known as an error or a bug. Therefore, failures (defined in Definition 4) can be occurred in design implementation or behavioral specification as a result of bugs or errors. There are different categories of bugs or errors that can be found in digital VLSI circuits but they can be divided into three main types: design errors, verification errors and manufacturing errors. Design errors also can be divided into: functional errors and electrical errors or circuit bugs. Functional errors are the functional inconsistences (or functional mismatches) in a design implementation as a result of incorrect wire connections or functional misbehavior of some gate elements. The main reason of occurring design errors is usually a designer interference during a synthesis phase in order to reach to a specific level of a system optimization. On the other hand, any bugs occurred in a behavioral specification is known as verification errors.

Definition 4: If the same primary input stimulus is applied to observe IC design with the same initial condition and at one or more observation points, there is an inconsistency between a design implementation (RTL abstraction) and its specification, it can be called as a *failure*.

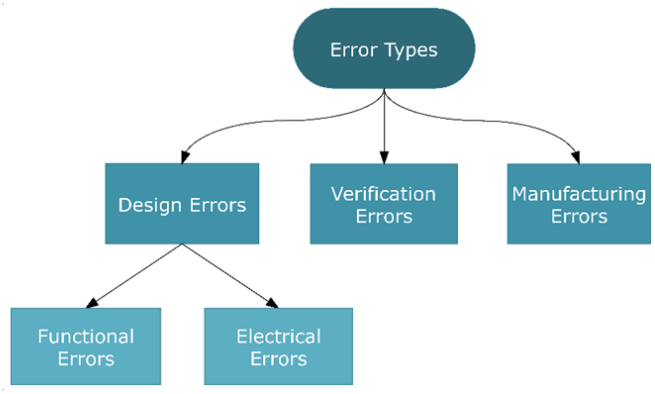


Figure. 1. Main Types of Errors or Bugs.

3.2. Bug Localization and Auto-Correction processes

In Bug localization process, counterexamples generated in logic verification phase are exploited to quickly detect potential locations of observed bugs. The efficient mechanism is identifying fault candidates by adding extra logic to faulty circuit and converting to a suitable formula to be handled by recent SAT solvers. Therefore, most debuggers take test patterns (logic assignments of inputs and their expected output of circuits) and buggy digital circuit then passed to four main stages as shown in figure 2 to finally detect fault candidates. The CNF formula [13, 14] of the

complete design can be easily refined and replicated to every test pattern and every time frame then passing to SAT solver to find error suspects

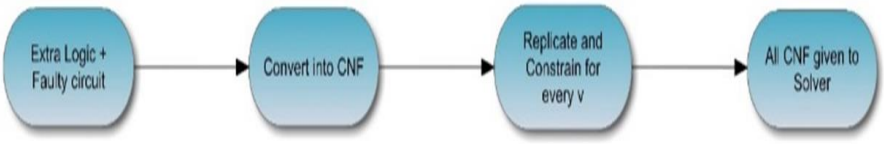
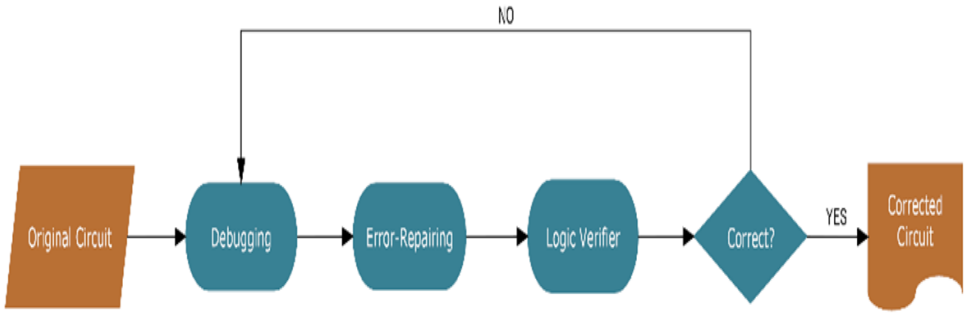


Figure 2. Main Steps of SAT-based Debugging Approach

In [15], a diagnosis technique is proposed to find potential bug locations using minimal correction subsets of UNSAT formula represented a faulty circuit. Also, the reasons of errors can be generated as a minimal unsatisfiable formulas (MUS) which can give designers which clauses combined together and produce the logic fault.

In most SAT-based auto-correction routines, the faulty circuit is automatically modified in order to be satisfy the given test pattern. Therefore, the accuracy and number of test patterns give to rectification algorithm have a high impact on the performance and accuracy of the rectified circuit. Therefore, the algorithms of auto-correction are classified into two main categories. Figure 3 demonstrates the main differences between those two categories. The first group of approaches as shown in figure (3.a) [16] focuses on the accuracy of corrected circuit by combining the debugger and correction algorithms together in order to achieve the equivalence between the actual circuit and the golden test patterns (as a black box of specification). Some techniques can be utilized to improve these error-fixing categories by abstracting and refining processes [17] for reducing the repetitive checking process. But these methods are not practical in a large-sized digital circuit as it has high consumed time and low resolution.

The second category is shown in figure 3.b. In this mechanism, the repetitive calling of logic debuggers and verifiers can be dispensed by fixing upon reduced counterexamples and using the original gate-level circuits [18]. Therefore, the performance of these techniques for rectifying design errors is improved but the accuracy is dependent on how many test patterns utilized.



a. First Mechanism

b. Second Mechanism

Figure 3. Categories of Debugging and Correction strategies.

4. THE PROPOSED RECTIFICATION ALGORITHM

In this section, the proposed automatic correction algorithm with generating compact test patterns (ACM-CTV) is described in more details.

Algorithm 1 illustrates the pseudo code of the proposed algorithm for generating both the Correct Circuit (CC) and Compact Test Vectors (CTV). The main inputs of this algorithm are:

1- Buggy circuit.
2- Buggy location.
3- Specification of the expected correct circuit.
4- Initial Test pattern (always one test pattern)

First of all, the Boolean values of inputs and outputs of buggy component should be defined. So, if it's one of the circuit's inputs and outputs, there is no need to call SAT solver. Otherwise, the buggy circuit with the initial test pattern are passed to SAT solver in CNF form in order to detect the Boolean values of inputs and outputs of the faulty location. Defining these assignments have a great impact on reducing the size of search space for generating the correct circuit and the compact TV. This process performed in lines 1-9 at Algorithm 1. Next, If the faulty gates have one input so the faulty circuit is enriched by 2-1 MUX. Otherwise, the faulty circuit is injected with 3-1 MUX. The inputs of 3-1 MUX are determined according to the detected Boolean values of inputs and output of faulty gate. These steps for generating CNF formula of MUX is performed in lines 10-16. After that lines 17-38, the steps for expecting the correct circuit with compact Tv are performed using the enriched circuit and the initial Tv. This process is a loop of generating two possible solutions (sol1, sol2), mitering between them, generating a new TV, pushing it into compact TV and retaining the possible correct solution as sol1 between two solutions for the next iteration. The loop stops when there is no second solution. Therefore, the sol1 is the correct component and the generated TV is the compact TV.

Algorithm 1: Pseudo code of Proposed ACM-CTV Method

Input: ECA_CTV ($Specs, CNF_b, BUGloc, Init_{tv}$):
Output: $CNF_c, Tv_{compact}$

1. $(TV_{Compact}, iter) \leftarrow (TV_{Init}, 1)$
2. $(Err_{gate}, Err_{Inp}) \leftarrow Find\_ErrorG(CNF_b, BUGloc)$
3. $(Prt, count) \leftarrow Check\_IO(Err_{gate})$
4. **if** $(Prt == 1 \;\&\&\; count = 2)$ **then**
5. $(I, O_{Correct}) \leftarrow Extract\_IOvalues(Init_{tv})$
6. **else**
7. $(I, O_{fault}) \leftarrow SAT\_Solver(CNF_b, Init\_tv)$
8. $O_{Correct} \leftarrow fliping\_func(O_{fault})$
9. **end-if**
10. **if** $(count == 1)$ **then** // NOT, BUFF or DFF gates
11. $MUX\_eq \leftarrow GEN\_2To1MUX(Err_{Inp})$
12. **else**
13. $PossMat \leftarrow Produce_{3DMatrix}()$
14. $Poss\_Gates \leftarrow Extract\_3Poss(I, O_{correct}, PossMat)$
15. $MUX\_eq \leftarrow Produce\_3To1\_MUX(Poss\_Gates)$
16. **end-if**
17. $CNF_e \leftarrow Enriched\_Func(CNF_b, MUX\_eq)$
18. $(cir1, sol1) \leftarrow GEN\_PossCorr(Poss\_Gates[0])$
19. **do**
20. **if** $(iter = 1)$ **then**
21. $(cir2, sol2) \leftarrow GEN\_PossCorr(Poss\_Gates[iter])$
22. **else**
23. $(cir2, sol2) \leftarrow Find\_OneSol(CNF_e, Sol1, Tv_{compact})$
24. **end-if**
25. $TV_{New} \leftarrow GEN\_Newtv(cir1, cir2, Specs)$
26. **if** $(Tv_{New} \neq \emptyset \;\&\&\; Tv_{New} \nexists Tv_{Compact})$ **then**
27. $Tv_{compact} \leftarrow Tv_{compact} \cup Tv_{New}$
28. **if** $(whichOne(cir1, Tv_{New}) == 0)$ **then** // cir1 is not compatible with Tv_{New}
29. $(cir1, sol1) \leftarrow (cir2, sol2)$
30. **end-if**
31. **else**
32. $CNF_e \leftarrow Block\_sol2(sol2)$
33. **end-if**
34. $iter++$
35. **while** $(sol2 = \emptyset)$
36. **end-do-while**
37. $CNF_c \leftarrow cir1$
38. ***return*** CNF_c

Algorithm 2 illustrates the pseudo code of the procedure of $Find\_OneSol$. This main goal of this process is to detect a probable second corrected circuit that can satisfy the generated compact test patterns and can be another solution than sol1. This process can be performed by duplicating the enriched circuit according to the current number (n) of compact test patterns then adding new constraints for:

1- representing compact test patterns
2- blocking solution 1.

Therefore, the final CNF instance can be a combination of duplicated circuits

$CNF_d = \sum_{i=0}^{n} CNF_e$, blocked clause $(cl_{blocked})$ and unit clauses for $Tv_{compact}$ as shown in equation 1, where :$n = Num\ of\ Tv_{compact}$

$$\varphi = cl_{blocked} \vee \sum_{i=0}^{n} CNF_e \vee UC_i$$

After creating equation 1, the solver is called by this instance to find a second solution (Sol2), the assigned values of MUX' selectors that can be used to produce cir2 which is used to find a new test pattern if it exists (line 25 in algorithm 1).

Algorithm 2: Find_OneSol Method

Input: $Find\_OneSol\ (CNF_e, Sol1, Tv_{compact})$:
Output: $cir, Tv_{compact}$

1. $CNF_d \leftarrow DUP\_Func(CNF_e, Err_{Inp})$
2. **for** $\forall tv_i \in Tv_{compact}$ **do**
3. $UC_i \leftarrow GPU\_UC(IMPd, tv_i)$
4. $CNF_d \leftarrow CombineCNF(CNF_d, UC_i)$
5. **end-for**
6. **if** $(Sol1 \neq \emptyset)$ **then**
7. $CNF_d \leftarrow Block\_sol1(CNF_d, sol1)$
8. **end-if**
9. $(st, \mu) \leftarrow Parallel\_Solver(CNF_d)$
10. **if** (st) **then**
11. $sol \leftarrow Extract\_Sol(\mu)$
12. **end-if**
13. $cir \leftarrow Gen\_Posscir\ (CNF_e, sol)$
14. ***return*** cir

Algorithm 3 illustrates the pseudo code of GEN_NewTv method that is used to generate new compact test pattern during searching for the correct circuit. This procedure has three main inputs:

1- Specification of the expected correct circuit.
2- The probable first circuit.
3- The probable second circuit.

This procedure is dependent on finding the Boolean values of inputs that can represent a difference between outputs of cir1 and cir2. For finding these values, a CNF instance represented a miter circuit between cir1 and cir2 is performed using $GEN_{miter}(cir1, cir2)$ (line 2). The miter circuit is nothing but XORs between the outputs of the circuit and OR between all XORs. After that, a new constraint is added to CNF_{mit} in order to reduce the search space that make the output of miter circuit as 1 (line 3). After that, this instance is passed to SAT solver. If the CNF formula is satisfied, the model returned from solver can be used to find the assigned values of inputs (Inp_{Tv})(line 6). Then, the expected correct output (Out_{Tv}) corresponding to Inp_{Tv} is generated using specification which can be combined with Inp_{Tv} (line 7) in one CNF formula and passed to SAT solver (line 8). Otherwise ($st = 0$), there is no new test pattern $Tv_{New} = \emptyset$. This means that Sol2 is a spurious solution (line 31-33 in algorithm1) that should be blocked in the enriched circuit before going to the next iteration in algorithm 1.

Algorithm 3: *GEN_NewTv* Method

Input: $GEN\_NewTv\ (cir1, cir2, Specs)$:
Output: Tv_{New}

1. $Tv_{New} \leftarrow \emptyset$
2. $CNF_{miter} \leftarrow GEN_{miter}(cir1, cir2)$
3. $CNF_{miter} \leftarrow Add\_Constraint(CNF_{miter}. output)$
4. $(st, \mu) \leftarrow Parallel\_Solver(CNF_{miter})$
5. **if** (st) **then**
6. $Inp_{Tv} \leftarrow Extract\_Inputs(\mu)$
7. $CNF_{specs} \leftarrow GEN\_fun(Specs, Inp_{Tv})$
8. $(st, \mu) \leftarrow Parallel\_Solver(CNF_{specs})$
9. $Out_{Tv} \leftarrow Extract\_Outputs(\mu)$
10. $Tv_{New} \leftarrow (Inp_{Tv}, Out_{Tv})$
11. **end-if**
12. ***return*** Tv_{New}

5. PERFORMANCE EVALUATION

In this section, a comparison between our proposed ACM-CTV algorithm with serial and parallel distribution and the previous proposed algorithm in [4] in terms of running time is proposed in table 1. The implementation is performed using ISCAS'85 benchmark [19] of SAT CNF instances (6 combinational circuits). Two algorithms were implemented in C++ and executed on Intel core i7-4510U working at 2.6 GHz with 8 GB system memory. As it proposed in compared algorithm in [4], every bug has been injected randomly in the Verilog module of a given circuit as a logic design error. Also, the SAT instance of every faulty digital circuit of ISCAS'85 is generated using a serial version of SAT encoder proposed in [13]. Also, the solving process is performed using Parallel CUD@SAT DPLL engine in two algorithms. Parallel subroutines are implemented in CUDA C and executed on NIVIDIA Geforce.

Table 1 illustrates the consumed time in seconds of previous correction process proposed in [4] and the proposed ACM-CTV correction algorithm. After analysis, the proposed ACM-CTV algorithm can rectify faulty digital circuit with full accuracy by generating compact test patterns during correction process in order to guarantee that there is no new test pattern can prove inconsistence. The proposed correction algorithm for single design errors in digital VLSI circuits delivers about 4.8x average speed compared to the latest existed correction method proposed in [4]. Also, the parallel version of test pattern distribution has a good benefit in case of using digital circuit with large ports as c432. Therefore, the maximum speed of parallel distribution of test patterns are 1.2x comparing to serial distribution proposed in [4].

Table 1. Comparison between running time (s) of previous Correction algorithm and proposed ACM-CTV algorithm.

CNF Type	#gates	#Inputs	#outputs	Time (s) of Correction Algorithm in [4]	Time(ms) of proposed ACM-CTV with serial dis.	Time (ms) of proposed ACM-CTV with parallel dis.
C17	6	5	2	0.420	0.018	0.469
C432	160	36	7	3.127	1.668	1.529
C499	202	41	32	3.195	2.654	3.038
C880	383	60	26	5.931	5.953	5.537
C1908	880	33	25	17.683	15.698	15.843
C3540	1699	50	22	40.389	31.356	34.445

6. CONCLUSIONS

In this paper, we propose an incremental auto-correction algorithm with generating compact test patterns. The main advantages of the proposed ACM-CTV algorithm are avoiding the dependency of the given test patterns by incrementally generating compact patterns and reducing the search space by injecting small size of in-circuit mutation (MUX 3x1 instead of MUX 6x1). By combining two enhanced methods, the proposed algorithm significantly outperforms the previous algorithm in [4] in terms of run time, delivering 4.8x average speed. Also, it shrinks the search space using small size of injected circuit and one test pattern in case of single faults. Therefore, the shrinking rate of search space is 6x compared to previous methods in [4, 11].

REFERENCES

[1] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-chip verification: methodology and techniques*: Springer Science & Business Media, 2007.
[2] J. Jaeger, "Virtually every ASIC ends up an FPGA," *EE Times*, 2007.
[3] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *International conference on theory and applications of satisfiability testing*, 2005, pp. 61-75.
[4] B. Alizadeh and Y. Abadi, "Incremental SAT-based Correction of Gate Level Circuits by Reusing Partially Corrected Circuits," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2020.
[5] A. Suelflow, G. Fey, R. Bloem, and R. Drechsler, "Using unsatisfiable cores to debug multiple design errors," in *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, 2008, pp. 77-82.
[6] Y. Chen, S. Safarpour, J. Marques-Silva, and A. Veneris, "Automated design debugging with maximum satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, pp. 1804-1817, 2010.
[7] L. G. Ali, A. I. Hussein, and H. M. Ali, "An efficient computation of minimal correction subformulas for SAT-based ATPG of digital circuits," in *Computer Engineering and Systems (ICCES), 2017 12th International Conference on*, 2017, pp. 383-389.
[8] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 1606-1621, 2005.
[9] K.-H. Chang, I. Wagner, I. Markov, and V. Bertacco, "Automatic error diagnosis and correction for RTL designs," ed: Google Patents, 2013.
[10] P. Behnam and B. Alizadeh, "In-circuit mutation-based automatic correction of certain design errors using SAT mechanisms," in *2015 IEEE 24th Asian Test Symposium (ATS)*, 2015, pp. 199-204.
[11] L. Gaber, A. I. Hussein, and M. Moness, "Improved Automatic Correction for Digital VLSI Circuits," in *2019 31st International Conference on Microelectronics (ICM)*, 2019, pp. 18-22.
[12] B. Alizadeh and S. R. Sharafinejad, "Incremental SAT-Based Accurate Auto-Correction of Sequential Circuits Through Automatic Test Pattern Generation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, pp. 245-252, 2018.
[13] M. Osama, L. Gaber, A. I. Hussein, and H. Mahmoud, "An Efficient SAT-Based Test Generation Algorithm with GPU Accelerator," *Journal of Electronic Testing*, vol. 34, pp. 511-527, 2018.
[14] L. G. Ali, A. I. Hussein, and H. M. Ali, "Parallelization of unit propagation algorithm for SAT-based ATPG of digital circuits," in *Microelectronics (ICM), 2016 28th International Conference on*, 2016, pp. 184-188.
[15] L. Gaber, A. I. Hussein, H. Mahmoud, M. M. Mabrook, and M. Moness, "Computation of minimal unsatisfiable subformulas for SAT-based digital circuit error diagnosis," *Journal of Ambient Intelligence and Humanized Computing*, 2020/06/29 2020.
[16] K.-H. Chang, I. L. Markov, and V. Bertacco, "Fixing design errors with counterexamples and resynthesis," in *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, 2007, pp. 944-949.
[17] S. Safarpour and A. Veneris, "Automated design debugging with abstraction and refinement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, pp. 1597-1608, 2009.

[18] A. Sulflow, G. Fey, C. Braunstein, U. Kuhne, and R. Drechsler, "Increasing the accuracy of SAT-based debugging," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009, pp. 1326-1331.

[19] D. Bryan, "The ISCAS'85 benchmark circuits and netlist format," *North Carolina State University*, vol. 25, 1985.