TEST Engineering & Management

January - February 2020
ISSN: 0193 - 4120 Page No. 13962 - 13967

Processor Testing Software by Implementing Rapid Self-test Technique

SantiSwarup Basa[1], Debashis Pradhan[2], Lipsa Das[3], Abhaya Kumar Panda[4], Santosh Kumar Swain[5]

[1,2,3]Department of Computer Science, North Orissa University, Baripada, Odisha, India
[4,5]Department of Computer Science Engineering, KIIT Deemed to be University, Bhubaneswar, India
cispubip@gmail.com

Article Info
Volume 82
Page Number: 13962 - 13967
Publication Issue:
January-February 2020

Article History
Article Received*: 18 May 2019*
Revised*: 14 July 2019*
Accepted*: 22 December 2019*
Publication*: 26 February 2020*

Abstract

Less Complex and rapid software-based self-test (SBST) methods are frequently used for evaluating contemporary processors due to the simplicity of synthesis using developmental methods, coverage for hard to check faults, non-invasive, small hardware overheads and alike. Moreover, the amount of time it takes for the SBST test synthesis is high. In this paper, an extension of the SBST method known as Rapid SBST (RSBST) is suggested; this decreases general sample synthesis speed by reprocessing simulation answers from current sample programs of the same observability. Test instructions, created expending the evolutionary process, which generates comparable fault simulation outcomes, are reprocessed for fault assessment. The proposed system utilizes the re-processing to increase the speed and promptness of the sample synthesis. The implemented RSBST methodology accelerates by an influence of "1.35" though preserving a fault of "1.35" and retaining a fault coverage above "96.2" percent.

Index Terms: *testing, software testing technique, RSBT, SBST.*

I. INTRODUCTION

As computer technology is complicated and growing, the accuracy of incorporated processors is very vital during the chip manufacturing and deployment phases. Classification of all physical faults has always been complicated since test patterns must be adapted to IC operating frequencies, which are exceptionally large. This at-speed test function is very hard to accomplish with internal tester techniques, as tester frequencies might not reach processor frequencies. To fix this problem, software-based self-test (SBST) methodologies [1], [2],[3] have developed software-based check scripts to be implemented to processors as sample protocols. These sample sequences are a series of orders with chosen operands that could affirm the features of the processor. SBST methods are non-intrusive even though the structure of the instrument does not require any change for experiments. SBST has subsequently evolved as an efficient solution or complementary technique for the production of microprocessors and integrated processors [4]–[8]and in-system testing. Key microprocessor businesses have acknowledged the ability of SBST to implement it in their sample streams. SBST is a non-intrusive strategy that integrates a "computer tester" with the shape of a self-test program in the on-chip space of the processor. As a result, SBST has enhanced flexible characteristics that can effectively address the issue of tiny storage array in-system screening that lacks MBIST. Software-Based Self-Test (SBST) has appeared as an efficient option for construction processors and in-system monitoring. For tiny storage applications that contain BIST circuitry, including cache label panels, SBST can be a versatile and low-cost alternative for the March experiment implementation and therefore a feasible complement to hardware methods. These less complex sample cards are uploaded to storage

January - February 2020
ISSN: 0193 - 4120 Page No. 13962 - 13967

places and the answers are accessed and contrasted for error detection. In addition, SBST does not involve any additional hardware that would result in a decreased sample price and null processor region penalty. This paper describes a powerful software self-test method called Rapid SBST (RSBST) where the experiment synthesis is quicker opposed to greedy oriented developmental methods and at the same moment does not reduce the fault coverage. In RSBST, redundant sample alternatives are recognized and their simulated outcomes are utilized for quicker fault coverage. This reusability system is designed to accelerate the traditional developmental sample synthesis.

II. RELATED WORK

Advances in the development of manual test patterns [9] have led significantly to the development of efficient testing programs. Intricate functional test patterns are created for the development of pipelined multi-threaded processors, vibrant command execution, and multi-core. A developmental method to automate the synthesis of auto software testing programs is described in the paper[10][11], [12]. The "μGP" self-adapting design is searching for improved sample programs. Earlier "μGP" methods utilize reporting coverage as a software usage metric. Later, the same method was expanded with "toggle, branch, expression and situation coverage"[13]. Furthermore, the testability of "corner-case faults" referred to as "hard-to-test faults" would be considerably low as they are not feasibly trackable by the synthesized test schemes. These prior "μGP" methods, could not ensure the performance of the experiment as these difficult-to-detect errors were kept unhindered and the code-based error assessment metrics do not have a rigorous connection with the gate-level error designs. These techniques could not achieve gate-level fault coverage of more than 90 percent for the synthesized match schemes[14]–[17]. These "μGP" tests progress in dozens of hours, which are fairly quick but require sufficient defect coverage. On the other side, the latest SBST automation approaches [18] have enhanced aspects of visibility but depend on effort-consuming sample production methods. Although the greedy element increases sample performance, this sample conception procedure [19][20], [21] would have to take roughly 170 hours of error assessment and contrast. Recommended in the classical SBST synthesis, the automation of processor experiment programs is carried out with the assistance of a developmental strategy that creates multiple experiment alternatives by employing genetic controllers. "SBST" strategies for target caches have also been described in some paper [22]. Almost all of them concentrate on converting "March algorithms" for in-system evaluation of caches at algorithmic stage or mitigating cost in terms of sample duration while maintaining "fault coverage"[23], whereas others continue providing pseudo-instruction patterns for "March activities" thorough experimentation outcomes of processor benchmarks[24][25]–[27].

III. PROPOSED WORK

In the suggested RSBST method, reprocessing of experiment response for similarly observable experiment programs is conducted to decrease the general sample production time. The general strategy of the RSBST processor automation system is shown in Figure. 1. In this system, the developmental core creates a sample alternative for optimum fault coverage that exploits the original population of sample schemes depicted as guided acyclic graphs (DAGs) and a training library. In the past, all the test systems were analyzed using an external evaluator. In order to decrease this expense of fault assessment, a computational complexity comparator is implemented, which contrasts and recognizes sample programs with comparable observability.

Published by: The Mattingley Publishing Co., Inc.

January - February 2020
ISSN: 0193 - 4120 Page No. 13962 - 13967

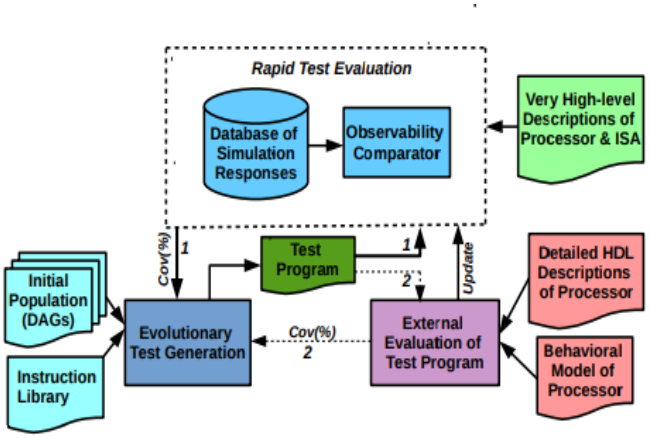


Figure 1 Block Diagram of Proposed System

Algorithm 1: Reusability of Greedy Coverage Method in RSBST Approach

Input: i^{th} generation of solutions $S_i^1, S_i^2, \ldots, S_i^{\mu+\lambda}$.
CF_i be the list of all covered faults until the generation i.
OBS_i^j be the values of observable destinations when S_i^j is executed.

Output: Modified CF_{i+1} of the $(i+1)^{th}$ generation.

1 If $OBS_i^j = OBS_i^{parent(j)}$, then the faut coverage list of S_i^j = the fault coverage list of $S_i^{parent(j)}$;
2 Else, evaluate the fault list of S_i^j using fault simulation.
3 The fitness function of S_i^j is $|F_j|$, which is the cardinality of the set of its newly covered faults, where F_j = fault coverage list of $S_i^j - CF_i$;
4 If the individual S_i^j is selected, update CF_{i+1} with CF_i and the newly detected faults: $CF_{i+1} = CF_i \cup F_j$;

The suggested technique of "reusability of fault simulation" findings is defined in Step 1 and Step 2 of the aforementioned Algorithm 1. In Figure.2, the paradigm for the proposed "rapid test evaluation" of Figure. 1, it's explained. Here, an elevated-level simulation sample program with a small timing necessity is used to obtain the components of all measurable places. Furthermore, the observability of the freshly created P testing program and its parent are contrasted to the extent of reusability.

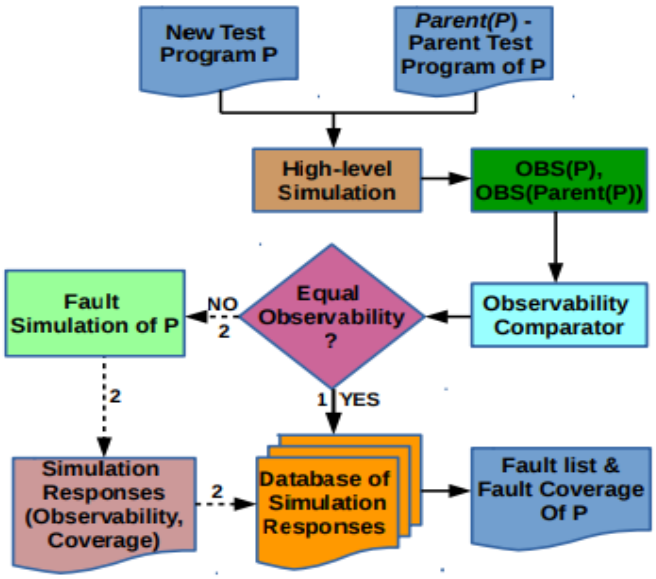


Figure 2 Test Generation in the proposed framework

IV. RESULTS

For the simulation of the proposed "RSBST test synthesis", similar kind of setup, benchmark processors, and fault models considered in is utilized. A 32-bit MIPS processor is also utilized and a "Leon3" processor model of "SPARC V8" architecture with a "7-stage pipeline" is also employed to be tested with the support of ten "behavioral fault representations" as shown in Table I.

Table 1 Behavioral fault representations

Fault Representation	Failure Type
Input stuck-at fault	Any primary input signal
Output stuck-at fault	Any primary output signal
If stuck then fault	If block is executed always
If stuck else fault	Else block is executed always
Elsif stuck then fault	Elsif block is executed always
Elsif stuck else fault	Elsif block is failed to get executed always
Assignment statement fault	Assignment of new values to a signal is failed always
Dead clause fault	A selected When clause in a case statement
Micro-operation fault	Micro-operations are failed always
Local stuck data fault	A signal object in a local expression

Published by: The Mattingley Publishing Co., Inc.

TEST Engineering & Management

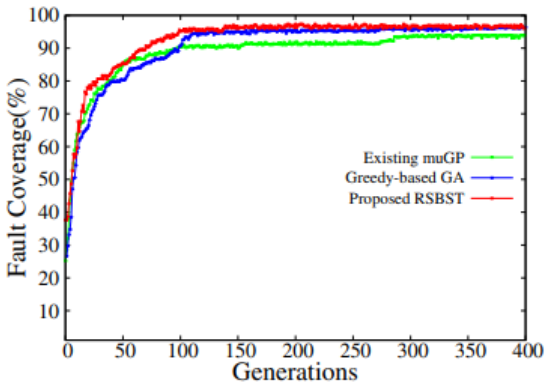
January - February 2020
ISSN: 0193 - 4120 Page No. 13962 - 13967

Table 2 Specifications utilized by proposed system

Specification	Value
Size of population (μ)	10
Number of offsprings (λ)	5
Number of generations	400
Selection methodology	Tournament selection
Size (τ) of the tournament	2
Steady-state threshold	40
Evolutionary methodology	ES($\mu + \lambda$) approach
Fault coverage evaluation method	Behavioral fault evaluation

The numbers of the parameters used for the suggested digital test are shown in Table II. The magnitude of the original sample "(μ)" is ten and the amount of offspring to be produced every generation is 5. The chromosomes of each batch are chosen by the "Tournament Selection Operator" in which the magnitude of the Tournament is 2. The developmental core performs a sample synthesis for four hundred generations and halts if there is no enhancement for forty generations, which is present in the steady-state limit.

The improvement in the growth of the sample synthesis using the suggested "RSBST" system is shown in Figure 4(a) for the "MIPS" processor. For the "Leon3" processor, the advancement of the error detection obtained using the "RSBST" system is shown in Figure 4 (b) of that. For the suggested "RSBST" strategy, the subsequent genes are replaced by likewise observable sibling cells together with the re-use of chosen DNA (μ= 10). Ultimately, the total reusable chromosomes would amount to 82.1 percent for the "MIPS" processor and 80.8 percent for the "Leon" processor.



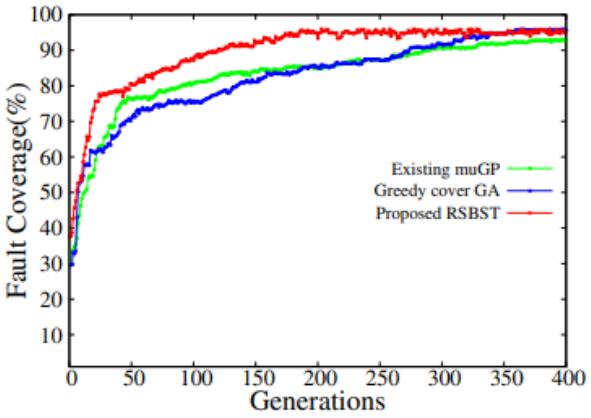


Figure 3 Outcomes of the simulation

V. CONCLUSION

In this research work, a quicker "SBST" processor fundamental synthesis is used by implementing an enhanced greedy-based developmental technique (RSBST) to develop sample schemes that could detect hard-to-detect faults. From the outcomes, consuming a more extensive fault model, the proposed approach is developing sample alternatives that could capture "97.3" per cent of the testable cognitive faults of the "MIPS" processor in "90" hours and "96.2" per cent of the sample failures of the "Leon3" processor in "98" hours. This confirms the genome (sample program) reuse of "82.1" percent for the "MIPS" processor and "80.8" percent for the "Leon3" processor. A quicker and reflective test synthesis may be implemented by utilizing the order of fragment reusability feature of the sample programs.

REFERENCES

[1] S. Agarwal and A. Bhatter, "Automated Software Test Data Generation Using Improved Search Procedure," Lect. Notes Softw. Eng., vol. 3, no. 2, pp. 152–156, 2015.

[2] H. Tahbildar and B. Kalita, "Automated Software Test Data Generation: Direction of Research," Int. J. Comput. Sci. Eng. Surv., vol. 2, no. 1, pp. 99–120, 2011.

[3] P. Sha'afiKabiri and Z. Navabi, "Effective RT-level software-based self-testing of embedded processor cores," in Proceedings of the 2012 IEEE 15th International Symposium on Design

January - February 2020
ISSN: 0193 - 4120 Page No. 13962 - 13967

and Diagnostics of Electronic Circuits and Systems, DDECS 2012, 2012, pp. 209–212.

[4] J. Tao and L. Chen, "Software test case automated generation algorithm with extended EDPN model," J. Networks, vol. 8, no. 8, pp. 1825–1830, 2013.

[5] C. Zhang, X. Bai, J. Li, and R. Zhang, "Automated test case generation for embedded software using extended interface automata," in Proceedings of the International Symposium on the Physical and Failure Analysis of Integrated Circuits, IPFA, 2013, pp. 292–298.

[6] Vishawjyoti and P. Gandhi, "A survey on prospects of automated software test case generation methods," in Proceedings of the 10th INDIACom; 2016 3rd International Conference on Computing for Sustainable Global Development, INDIACom 2016, 2016, pp. 3867–3871.

[7] M. KaurBrar and S. Garg, "Survey on Automated Test Data Generation," Int. J. Comput.Appl., vol. 108, no. 15, pp. 1–4, 2014.

[8] D. Wei, L. Tang, X. Li, and L. Shang, "Research on Automated Software Test Case Generation," J. Softw., vol. 9, no. 11, 2014.

[9] M. De Carvalho, P. Bernardi, E. Sanchez, M. S. Reorda, and O. Ballan, "Increasing the fault coverage of processor devices during the operational phase functional test," J. Electron.Test. Theory Appl., 2014.

[10] B. Baudry, S. Ghosh, F. Fleurey, R. France, Y. Le Traon, and J. M. Mottu, "Barriers to systematic model transformation testing," Commun. ACM, 2010.

[11] S. Anand et al., "An orchestrated survey of methodologies for automated software test case generation," J. Syst. Softw., 2013.

[12] K. V. Hanford, "Automatic generation of test cases," IBM Syst. J., 2010.

[13] J. V. Millo, S. Ramesh, S. N. Krishna, and G. K. Narwane, "Compositional verification of software product lines," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2013.

[14] A. Riefert, R. Cantoro, M. Sauer, M. SonzaReorda, and B. Becker, "A Flexible Framework for the Automatic Generation of SBST Programs," IEEE Trans. Very Large Scale Integr.Syst., 2016.

[15] A. Riefert, R. Cantoro, M. Sauer, M. S. Reorda, and B. Becker, "Effective generation and evaluation of diagnostic SBST programs," in Proceedings of the IEEE VLSI Test Symposium, 2016.

[16] P. A. Ivanenko and A. Y. Doroshenko, "Method of Automated Generation of Autotuners for Parallel Programs," Cybern. Syst. Anal., 2014.

[17] F. Martinelli and I. Matteucci, "A framework for automatic generation of security controller," Softw. Test. Verif. Reliab., 2012.

[18] A. Riefert, R. Cantoro, M. Sauer, M. S. Reorda, and B. Becker, "On the automatic generation of SBST test programs for in-field test," in Proceedings -Design, Automation and Test in Europe, DATE, 2015.

[19] V. M. Suryasarman, S. Biswas, and A. Sahu, "Automation of Test Program Synthesis for Processor Post-silicon Validation," J. Electron.Test. Theory Appl., 2018.

[20] P. Moharikar and J. Guddeti, "Automated test generation for post silicon microcontroller validation," in 2017 IEEE International High Level Design Validation and Test Workshop, HLDVT 2017, 2017, vol. 2017-January, pp. 45–52.

[21] M. Dehbashi and G. Fey, Debug automation from pre-silicon to post-silicon. 2015.

[22] A. Karputkin and J. Raik, "A synthesis-agnostic behavioral fault model for high gate-level fault coverage," in Proceedings of the 2016 Design, Automation and Test in Europe Conference and Exhibition, DATE 2016, 2016.

[23] S. Di Carlo, P. Prinetto, and A. Savino, "Software-based self-test of set-associative cache memories," IEEE Trans. Comput., 2011.

[24] G. Theodorou, S. Chatzopoulos, N. Kranitis, A. Paschalis, and D. Gizopoulos, "A Software-Based Self-Test methodology for on-line testing of data TLBs," in Proceedings - 2012 17th IEEE European Test Symposium, ETS 2012, 2012.

[25] N. Duong, T. Kim, D. Zhao, and A. V. Veidenbaum, "Revisiting level-0 caches in embedded processors," in CASES'12 - Proceedings of the 2012 ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems, Co-located with ESWEEK, 2012.

Published by: The Mattingley Publishing Co., Inc.

January - February 2020
ISSN: 0193 - 4120 Page No. 13962 - 13967

[26] G. Theodorou, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Software-based self-test for small caches in microprocessors," IEEE Trans. Comput. Des.Integr. Circuits Syst., 2014.
[27] G. Theodorou, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Software-based self test methodology for on-line testing of l1 caches in multithreaded multicore architectures," IEEE Trans. Very Large Scale Integr. Syst., 2013.

Published by: The Mattingley Publishing Co., Inc.