Effective techniques for automatically improving the transition delay fault coverage of Self-Test Libraries

Effective techniques for automatically improving the transition delay fault coverage of Self-Test Libraries

Riccardo Cantoro*, Francesco Garau*, Patrick Girard†, Nima Kolahimahmoudi*,
Sandro Sartoni*, Matteo Sonza Reorda* and Arnaud Virazel†

*Department of Computer and Control Engineering
Politecnico di Torino
Turin, Italy

†LIRMM
University of Montpellier / CNRS
Montpellier, France

***Abstract*—In-field test of integrated circuits using Self-Test Libraries (STLs) is a widely used technique specifically suited to guarantee the processor's correct behavior during the operative lifetime, as mandated by functional safety standards such as ISO26262. Developing STLs for stuck-at faults requires significant manual efforts from test engineers, and targeting delay faults is even more challenging. In order to support this process, in this paper we propose a method to automate the creation of STLs targeting delay faults starting from existing STLs targeting stuck-at faults. The method is based first on identifying excited but not-observed transition delay faults and then adding suitable instructions able to detect them. Experimental results on a RISC-V processor show that the method can systematically detect a significant percentage of the target faults with reasonable computational effort and test code size increase.**

***Index Terms*—software-based self-test, software test libraries, in-field test, safety, functional test, delay faults**

I. Introduction

New semiconductor technologies are increasingly adopted in emerging applications. Such technologies, however, are extremely complex and sophisticated, leading to more frequent physical defects and reduced operative lifetime. Most of these defects are tested by targeting delay faults, such as transition delay faults (TDFs). Functional testing is an effective solution to tackle such issues. In the form of Software-Based Self-Test (SBST), functional testing [1] is based on the execution of STLs by the device under test (DUT). This approach has been proved effective both when processor cores [2]–[6] and peripherals [7]–[9] are tested. SBST is a desirable solution for *in-field testing*, i.e., when the device's reliability and safety has to be guaranteed throughout the operative lifetime, and can be used whenever compliance to standards such as the *ISO26262 standard* for automotive systems is required.

Developing STLs from scratch, however, is not trivial, as it requires a significant amount of manual effort. Moreover, understanding why certain faults are not detected is not always easy. The work in [10] moves the first step in classifying not-observed TDFs, defining some upper boundaries on how much the final TDF coverage can be increased.

In this paper, we propose an automatic and systematic methodology to increase the TDF coverage of STLs by detecting faults identified in [10] on complex pipelined processor cores, starting from a set of test programs devised for stuck-at faults (SAFs). The reported results show that it is possible to detect most of the TDFs (increasing the coverage by up to 15%) with a reasonable test time increase (from 15% to 22%) and computational effort. The article is organized as follows: in Section II related works are presented, while in Section III we describe the proposed approach. In Section IV we present the experimental results and, finally, in Section V we draw the conclusions.

II. Background

[10] introduces a study on TDFs in modern pipelined CPUs that have not been observed throughout the execution of STLs targeting SAFs. The article introduces *User Accessible Register* faults (UARs), whose effects reached registers that can be directly observed through instructions from the CPU's ISA, and *Hidden Register* (HRs) faults, whose effects reached registers non directly controllable. [10] provides some useful insights on what faults to target to improve the final fault coverage, and an upper bound on how much the final transition delay fault coverage can be improved. Strategies to detect them, however, are not given. Articles like [11], [12] focus on the improvement of available programs to obtain high fault coverage figures. These works show that methodologies for improving test programs can be successfully devised. Nonetheless, they are developed bearing the classical stuck-at fault model in mind.

III. Proposed Approach

This work focuses on the UAR fault category introduced in [10], defining some internal observation points, that do not require any hardware modification, to be used during simulation to further refine the topological analysis about fault effects.

In order to detect UAR faults, first we analyze the fault data base produced at the end of the fault simulation where internal observation points have been added. This is done to understand what UAR we have to work on, and what portion of the test program should be improved. In modern in-order pipelined CPUs there are instructions that take more than one

clock cycle to go through the CPU execution stage (*multi-cycle instructions*), and *single-cycle instructions* that only take one cycle in the execution stage. The former in particular should be carefully taken into account, as the fault effect might be overwritten during the required execution cycles. When dealing with *single-cycle instructions*, a *store* operation on the register affected by the faulty value after the time at which said effect reached the register, and before the register is overwritten by another operation, is sufficient. *Multi-cycle instructions*, instead, can be dealt with in two ways: if the fault effect is still present at the last execution cycle the same strategy adopted for *single-cycle instructions* is used, else we modify the operands of the multi-cycle instructions to ensure that the faulty value reaches the register towards the end of the execution stage. As demonstrated by the experimental results, identifying suitable operands for this purpose is a feasible task, which can often performed following a try-and-error approach.

Finally, it is noted that a set of instructions added to the original STL may be capable of detecting more than one TDF at the same time, thus achieving better fault coverages with a smaller test program. This is only possible for all those TDFs whose effects propagate to the same register at the same time.

IV. Experimental Results

A. Case study

This work has been validated on PULPino [13], a 32-bit RISC-V-based SoC platform developed by ETH Zurich and Università di Bologna. We adopted three different STLs that were originally intended to test SAFs on the PULPino core. In order to ensure a diverse and realistic testbench, the three test programs we selected have been developed following different implementation strategies. Fault simulations have been carried out using Synopsys Z01X, a commercial tool devised specifically for functional safety. The full flow of STL improvement and fault simulation for TDFs took no longer than 4 days on an Intel Xeon CPU E5-2680 v3 server with a clock frequency up to 3.3GHz.

B. Achieved results

Table I shows data on the UAR faults that were detected as a result of the proposed methodology.

TABLE I
Analysis on detected UAR faults

	STL1	STL2	STL3
Detected UARs	6,578	23,912	2,864
Total UARs	6,591	23,922	2,900
%Detected UARs	99.80	99.96	98.76
Code size [kB]	6.34	4.17	3.53

Looking at the table, our approach is capable of detecting almost every fault out of those that are excited but not detected by the existing STL, with the worst case scenario being STL3 with a 98.76% of UAR faults being detected. Given a total amount of 159,326 transition delay faults, through our methodology we can increase the final fault coverage by 4.13% for STL1, 15.01% for STL2, and 1.80% for STL3, respectively. This improvement comes with an increase of the final code size, which amounts to an additional 22.21% for STL1, 14.97% for STL2, and 21.16% for STL3. This proves that our strategy is able to systematically test not-observed transition delay faults whose effects reached user accessible registers.

V. Conclusions

This work introduces an automated and systematic methodology to detect TDFs whose effects have been excited but not observed by already available STLs. Starting from a library of STLs developed for SAFs, our approach defines strategies to detect transition delay faults belonging to the class of UAR faults. Experimental results gathered on a RISC-V test case show that we are able to detect almost every fault affecting UARs. Such increase in fault coverage comes with a reasonably small increase of the code size, with the worst case scenario consisting in about 22% added code size. Future works will include the definition of effective strategies to test HR faults, in order to match as closely as possible the upper bounds in recoverable fault coverage presented in [10].

References

[1] M. Psarakis *et al.*, "Microprocessor Software-Based Self-Testing," *IEEE Design & Test of Computers*, vol. 27, no. 3, pp. 4–19, 2010.

[2] K. Christou *et al.*, "A Novel SBST Generation Technique for Path-Delay Faults in Microprocessors Exploiting Gate- and RT-Level Descriptions," in *IEEE VTS*, April 2008, pp. 389–394.

[3] V. Singh *et al.*, "Instruction-Based Self-Testing of Delay Faults in Pipelined Processors," *IEEE Transactions on VLSI Systems*, vol. 14, no. 11, pp. 1203–1215, Nov 2006.

[4] P. Bernardi *et al.*, "Development Flow for On-Line Core Self-Test of Automotive Microcontrollers," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 744–754, 2016.

[5] Wei-Cheng Lai *et al.*, "Test program synthesis for path delay faults in microprocessor cores," in *IEEE ITC*, 2000, pp. 1080–1089.

[6] P. Bernardi *et al.*, "A Deterministic Methodology for Identifying Functionally Untestable Path-Delay Faults in Microprocessor Cores," in *International Workshop on MTV*, Dec 2008, pp. 103–108.

[7] M. Grosso *et al.*, "Software-Based Self-Test for Transition Faults: a Case Study," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2019, pp. 76–81.

[8] R. Cantoro *et al.*, "In-field functional test of can bus controllers," in *IEEE VTS*, 2020, pp. 1–6.

[9] A. Apostolakis *et al.*, "Test Program Generation for Communication Peripherals in Processor-Based SoC Devices," *IEEE Design & Test of Computers*, vol. 26, no. 2, pp. 52–63, 2009.

[10] R. Cantoro *et al.*, "Self-test libraries analysis for pipelined processors transition fault coverage improvement," in *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2021, pp. 1–4.

[11] A. Ruospo *et al.*, "On-line Testing for Autonomous Systems driven by RISC-V Processor Design Verification," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2019, pp. 1–6.

[12] A. Jasnetski *et al.*, "Automated software-based self-test generation for microprocessors," in *International Conference MIXDES*, 2017, pp. 453–458.

[13] ETH Zurich and Università di Bologna, "PULPino microcontroller system." [Online]. Available: https://github.com/pulp-platform/pulpino