Self-Test Libraries Analysis for Pipelined Processors Transition Fault Coverage Improvement

Self-Test Libraries Analysis for Pipelined Processors Transition Fault Coverage Improvement

Riccardo Cantoro*, Patrick Girard†, Riccardo Masante,*
Sandro Sartoni*, Matteo Sonza Reorda*, Arnaud Virazel†

*Politecnico di Torino
Turin, Italy

†LIRMM
University of Montpellier / CNRS
Montpellier, France

***Abstract*—Testing digital integrated circuits is generally done using Design-for-Testability (DfT) solutions. Such solutions, however, introduce non-negligible area and timing overheads that can be overcome by adopting functional solutions. In particular, functional test of integrated circuits plays a key role when guaranteeing the device's safety is required during the operative lifetime (*in-field test*), as required by standards like ISO26262. This can be achieved via the execution of a Self-Test Library (STL) by the device under test (DUT). Nevertheless, developing such test programs requires a significant manual effort, and can be non-trivial when dealing with complex modules. This paper moves the first step in defining a generic and systematic methodology to improve transition delay faults' observability of existing STLs. To do so, we analyze previously devised STLs in order to highlight specific points within test programs to be improved, leading to an increase in the final fault coverage.**

***Index Terms*—software-based self-test, software test libraries, on-line test, transition delay test, safety, functional test**

I. Introduction

New advanced semiconductor technologies are increasingly adopted in emerging applications. Such technologies, however, are extremely complex and sophisticated, leading to more frequent physical defects and a reduced operative lifetime. Testing integrated circuits (ICs), hence, is of paramount importance. Some of these defects are tested by targeting delay faults, i.e., faults that affect the timing behavior of the device under test (DUT), such as transition delay faults (TDFs) or path delay faults (PDFs).

Testing integrated circuits is usually done through Design-for-Testability (DfT) solutions, which require the addition of hardware modules, such as Logic BIST or scan chains. Although based on mature technology and supported by most EDA tools, DfT imposes non-negligible timing and area overheads that could degrade performances. Moreover, functionally untestable faults [1] (FUFs), i.e., faults whose effects are never observed within functional scenarios, will possibly be detected, lowering the yield. These issues can be overcome by adopting functional testing. In the form of Software-Based Self-Test (SBST), functional testing [2], [3] is based on the execution of Self-Test Libraries (STLs) by the DUT. This approach has been proved effective both when processor cores [4]–[11] or peripherals [12]–[15] are tested, and several companies provide STLs for their products [16]–[19]. SBST is a common solution for *in-field testing*, i.e., when the device's reliability and safety has to be guaranteed throughout the operative lifetime, and it is also cheap and flexible. Thanks to this, SBST can be successfully used whenever compliance to safety standards such as *ISO26262* is required.

However, developing STLs from scratch is not a trivial task and requires a non-negligible amount of manual effort, more so when targeting complex devices: test programs must be able to excite as many faults as possible and make their effects observable at primary outputs (POs) by using instructions from the system's instruction set, only [20]. Moreover, understanding why certain faults are not detected is not always easy. This paper moves the first step in defining a generic and systematic flow capable of improving TDF coverage on complex pipelined processor cores, starting with a set of test programs devised for stuck-at faults (SAFs). We choose the transition delay fault model over the path delay one because TDFs are much better supported by both standards and EDA tools. The main contributions of this work are the definition of internal observation points, to better understand where the effects of not detected faults propagated and stopped, and the creation of a software simulation trace, to understand which instruction blocks within the STL can excite and propagate faults. This approach is validated on a RISC-V core, using available commercial tools and a set of pre-existing STLs targeting SAFs. The reported results show that it is possible to identify several easy-to-perform changes in the STLs and introduce an increase in the TDF Fault Coverage up to nearly 18%.

The article is organized as follows: in Section II a background on related works is outlined, 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

Test programs development for delay faults through an SBST approach is a well-studied topic in academia [6], [8], [10]. Regarding TDFs specifically, [12] describes STL development strategies for peripherals embedded in System on Chips, achieving significant fault coverage figures. Work [4] describes a methodology to test delay faults on computational

blocks within superscalar processors, while [5] aims at testing RISC-like CPUs by dividing them into modules under test and devising test strategies for each of these modules. Finally, [21] presents a reinforcement learning-based test program generation technique for TDFs validated on a MIPS32 core. Although effective, these works require the generation of test programs starting from scratch. [21], moreover, requires the usage of a reinforcement learning algorithm, which might not be effective when tackling high-complexity cores.

Articles [22]–[24] focus on the improvement of available programs to obtain high fault coverages. [22] describes how to derive test patterns for online testing starting from programs originally intended for verification purposes, significantly increasing the final coverage of stuck-at faults on a RISCV core. Works [23], [24] present a tool based on High-Level Decision Diagrams for modeling microprocessors and faults, used in conjunction with previously prepared code templates to generate the final self-test program. These works show that methodologies for improving test programs, although tailored for SAFs, can be successfully devised. However, the manual effort for developing test programs implementing the above techniques can be relevant.

III. Proposed Approach

In this paper, we introduce a systematic methodology that, given test programs already developed for SAFs, allows to pinpoint code regions within the STL to be modified to increase transition delay fault coverage in complex pipelined processor cores, as shown in Fig. 1. The reason for doing this is twofold: not only methodologies for developing effective SAF-oriented STLs are already available and capable of achieving high fault coverages, but the stuck-at and transition delay fault models also share some similarities in the way they are tested. Detecting a slow-to-rise (STR) fault implies detecting the relative stuck-at-0 fault, and similarly for slow-to-fall (STF) and stuck-at-1 faults. In this work, we take SAF-oriented test programs as our starting point: for this reason, SAF coverages are used as a reference value for the TDF ones.

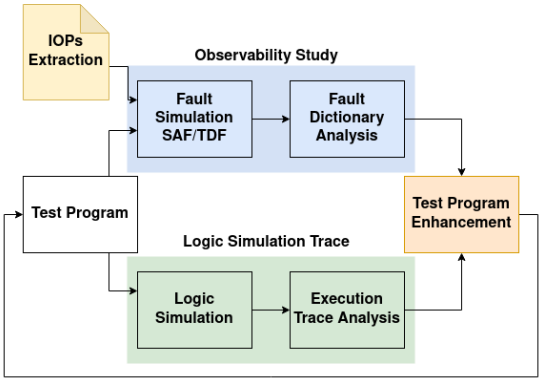

Fig. 1. Proposed test flow

A. Observability Study

This process aims to give some insights on undetected faults, divided into not controlled (NC) and not observed (NO) faults. Devising test strategies for such faults not only depends on whether they were NC or NO but also, in the latter case, where their effects propagated and stopped. For this reason, we define two groups of internal observation points (IOPs), namely *User Accessible Registers (UARs)*, i.e., registers that the user can directly access through available instructions, and *Hidden Registers (HRs)*, i.e., registers hidden within submodules or glue logic that the user cannot directly access. For example, general-purpose registers (GPRs) or control status registers (CSRs) are UARs, while pipeline registers are HRs. To easily identify which IOP a fault has reached during the fault simulation, we also introduce *user-defined status labels*. This information is then used in the subsequent two fault simulations, targeting both stuck-at and transition delay faults. While performing such simulations, the test flow records additional data into a *fault dictionary*, including at what time instant faults have been observed at POs (i.e., detected faults) or at one of the aforementioned IOPs, together with good and faulty machine values for comparison. Together with the fault dictionary, the observability study allows to analyze and correlate SAF and TDF coverages. We divide stuck-at and transition delay faults into three categories: NC, NO, and DT (detected), and we build a fault report table that can be divided into three main sections as follows:

1) *NC stuck-at faults*: contains all stuck-at faults not controlled by the test program. A not controlled SAF implies the corresponding TDF is not controlled as well; hence, in this section, we also include the subsequent NC TDFs.
2) *NO stuck-at faults*: encompasses all not-observed SAFs. When a SAF is not observed, the relative TDF might be either NO or NC. In the former case, we show all NO TDFs dividing them into faults that reached UARs, those that reached HRs, and those that didn't reach any sequential element (NS). In the latter case, we repeat the same analysis on SAFs.
3) *DT stuck-at faults*: this section includes all detected SAFs. TDFs related to detected SAFs might either be NC, NO, or DT, so we report all three sub-categories, repeating the IOPs analysis for NO TDFs.

B. Logic Simulation Trace

With the Logic Simulation Trace process, we map the execution time to the instructions currently executed by the processor core. To do so, a module to be attached to the RTL description of the DUT is required. This module is activated when the test program's logic simulation is launched and closely monitors internal signals like the clock, the program counter (PC), and the instruction's opcode coming from the instruction RAM. During the simulation, the tracer stores the monitored values at each clock cycle, also looking for both source and destination registers values from the decode and the execution stage. Combining data from the fault dictionary and the execution trace, the proposed test flow allows to easily identify what portion of the code must be improved to cover NO faults, as shown in Fig. 2. Which and how many

instructions to use in general depends on the IOP reached by the fault and is not the main focus of the current paper.

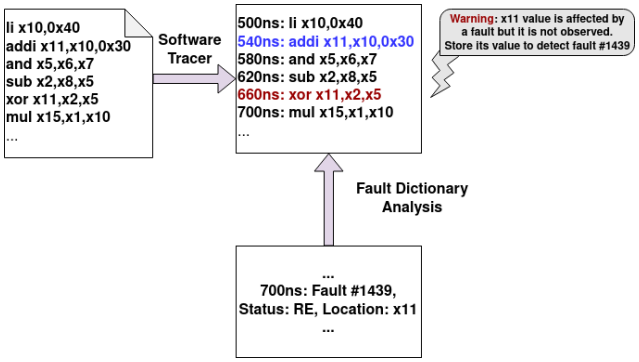

Fig. 2. STL improvement strategy

IV. Experimental Results

A. Case study

The approach presented in this work has been validated on PULPino [25], an open-source single-core SoC platform based on a 32-bit RISC-V core developed by ETH Zurich and Università di Bologna. This core has been synthesized using the 45nm Silvaco Open Cell library [26], setting the clock period to 40ns, resulting in a total amount of $46,850$ gates, a total area (eq. gates) of $51,001.65$, and $159,326$ transition delay faults. As for the test programs, we decided to start with three STLs, developed following different implementation strategies to test SAFs on the PULPino core, namely STL1, STL2, and STL3. A summary of the execution time (expressed in the total amount of clock cycles), memory size, and SAF coverage is reported in Table I.

TABLE I
STLs general information

Test Program	*#Clock cycles*	*Memory size [kB]*	*SAF coverage %*
STL1	17,308	27.32	81.42
STL2	31,158	27.86	81.86
STL3	80,455	16.68	82.18

All fault simulations have been performed using Synopsys Z01X, a commercial tool devised specifically for functional safety, with the advantage of including user-defined status labels, and having a quick fault simulation. Performing stuck-at and transition delay fault simulations took, for each test program, no longer than 18 hours on an Intel Xeon CPU E5-2680 v3 server with a clock frequency up to 3.3GHz.

B. Fault simulation results

Since the simulation trace step is quite straightforward, we focus on the data obtained by fault simulating the three STLs. In Table II we report the fault report table introduced in Section III-A for the whole processor core. For each column, we show the total amount of faults belonging to each category and, in the final section, the total amount of faults together with the stuck-at and transition delay fault coverages.

TABLE II
SAF and TDF classification for the adopted STLs

Fault Categories	#Faults STL1	#Faults STL2	#Faults STL3
SAF_NC	15,260	15,018	14,572
TDF_NC	15,260	15,018	14,572
SAF_NO	14,346	13,885	13,826
TDF_NO	10,992	10,882	10,801
TDF_UAR	545	587	410
TDF_HR	5,061	3,758	5,043
TDF_NS	5,386	6,537	5,348
TDF_NC	3,354	3,003	3,025
SAF_UAR	615	307	338
SAF_HR	2,399	2,307	2,349
SAF_NS	340	389	338
SAF_DT	129,720	130,423	130,928
TDF_DT	98,356	69,875	99,310
TDF_NO	8,975	24,088	8,822
TDF_UAR	2,598	6,860	2,484
TDF_HR	6,376	17,228	6,338
TDF_NS	1	0	0
TDF_NC	22,389	34,640	22,796
TOTAL	159,326	159,326	159,326
SAF Coverage %	81.42	81.86	82.18
TDF Coverage %	61.73	44.19	62.54

Let us focus on *TDF_UAR* faults from blocks 2 (*SAF_NO*) and 3 (*SAF_DT*). These faults have reached some user-accessible registers like the register file or control status registers. Thus, detecting them is fairly easy once the right register and time instant are identified, which can be easily done through the logic simulation trace process. Detecting faults from this category leads to an increase in TDF coverage of 2 percentile units (p.u.) for STL1, 4.7 for STL2, and 1.8 for STL3. To these faults, we have to add the contribution of *TDF_HR* faults, which need some more complex strategies to let them reach a PO (or a UAR, after which a store is sufficient). If we factor in their contribution, the TDF coverage can be increased by 9.15 p.u. for STL1, 8.96 for STL2, and 17.85 for STL3. In addition to that, although this is not the goal of this work, covering *TDF_UAR* and *TDF_HR* from the second block allows covering the relative SAFs as well, since a detected TDF implies the relative SAF detected as well.

V. Conclusions

In this work, we presented a generic and systematic flow to help the test engineer easily identify regions in STLs to be improved to increase TDF coverages on complex pipelined processor cores. Although devising strategies to cover such faults is not the goal of this paper, we show that, given the right strategies, it is possible to enhance the TDF coverage from 9 up to — if an STL is not optimized for TDF — 17.85 percentile units. Future works will focus on the aforementioned strategies to cover transition delay faults whose effects stopped in hidden registers.

REFERENCES

[1] P. Bernardi *et al.*, "On-line functionally untestable fault identification in embedded processor cores," in *Design, Automation & Test in Europe Conference Exhibition (DATE)*, 2013, pp. 1462–1467.

[2] M. Psarakis *et al.*, "Systematic software-based self-test for pipelined processors," in *ACM/IEEE Design Automation Conference (DAC)*, 2006, pp. 393–398.

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

[4] N. Hage *et al.*, "On Testing of Superscalar Processors in Functional Mode for Delay Faults," in *International Conference on VLSI Design and International Conference on Embedded Systems (VLSID)*, 2017, pp. 397–402.

[5] A. S. Oyeniran *et al.*, "Implementation-Independent Functional Test for Transition Delay Faults in Microprocessors," in *Euromicro Conference on Digital System Design (DSD)*, 2020, pp. 646–650.

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

[7] C. H. . Wen *et al.*, "On a software-based self-test methodology and its application," in *IEEE VLSI Test Symposium (VTS)*, 2005, pp. 107–113.

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

[9] 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.

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

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

[12] 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.

[13] R. Cantoro *et al.*, "In-field functional test of can bus controllers," in *IEEE VLSI Test Symposium (VTS)*, 2020, pp. 1–6.

[14] 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.

[15] A. van de Goor *et al.*, "Memory testing with a RISC microcontroller," in *Design, Automation & Test in Europe Conference Exhibition (DATE)*, 2010, pp. 214–219.

[16] Hitex, "Microcontroller self-test libraries." [Online]. Available: https://www.hitex.com/tools-components/software-components/selftest-libraries-safety-libs/pro-sil-safetlib/

[17] ARM, "Enabling Our Partnership to Bring Safer Solutions to the Market Faster." [Online]. Available: https://developer.arm.com/technologies/functional-safety

[18] Microchip Technology Inc., "16-bit CPU Self-Test Library User's Guide," 2012. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/52076a.pdf

[19] STMicroelectronics, "Guidelines for obtaining IEC 60335 Class B certification for any STM32 application," Mar 2016. [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/application_note/02/1a/91/78/e4/15/4d/35/CD00290100.pdf/files/CD00290100.pdf/jcr:content/translations/en.CD00290100.pdf

[20] J. Perez Acle *et al.*, "Observability Solutions for In-Field Functional Test of Processor-Based Systems," *Microprocessors and Microsystems*, p. 392–403, 2016.

[21] C. Y. Chen *et al.*, "Reinforcement-Learning-Based Test Program Generation for Software-Based Self-Test," in *IEEE Asian Test Symposium (ATS)*, 2019, pp. 73–735.

[22] 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.

[23] A. Jasnetski *et al.*, "On automatic software-based self-test program generation based on high-level decision diagrams," in *IEEE Latin-American Test Symposium (LATS)*, 2016, pp. 177–177.

[24] ——, "Automated software-based self-test generation for microprocessors," in *International Conference Mixed Design of Integrated Circuits and Systems (MIXDES)*, 2017, pp. 453–458.

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

[26] Silvaco, "Silvaco 45nm open cell library." [Online]. Available: https://www.silvaco.com/products/nangate/FreePDK45_Open_Cell_Library/