Microprocessors and Microsystems 000 (2016) 1–12

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Observability solutions for in-field functional test of processor-based systems: A survey and quantitative test case evaluation

J. Perez Acle[a,*], R. Cantoro[b], E. Sanchez[b], M. Sonza Reorda[b], G. Squillero[b]

[a] *Universidad de la República, Montevideo, Uruguay*
[b] *Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy*

A R T I C L E I N F O

Article history:
Received 1 April 2016
Revised 24 July 2016
Accepted 10 September 2016
Available online xxx

Keywords:
Functional test
Software-based self-test
Performance counters
Fault simulation
Observability

A B S T R A C T

The usage of electronic systems in safety-critical applications requires mechanisms for the early detection of faults affecting the hardware while the system is in the field. When the system includes a processor, one approach is to make use of functional test programs that are run by the processor itself. Such programs exercise the different parts of the system, and eventually expose the difference between a fully functional system and a faulty one. Their effectiveness depends, among other factors, on the mechanism adopted to observe the behavior of the system, which in turn is deeply affected by the constraints imposed by the application environment. This paper describes different mechanisms for supporting the observation of fault effects during such in-field functional test, and it reports and discusses the results of an experimental analysis performed on some representative case studies, which allow drawing some general conclusions. The gathered results allow the quantitative evaluation of the drop in fault coverage coming from the adoption of the alternative approaches with respect to the ideal case in which all the outputs can be continuously monitored, which is the typical scenario for test generation. The reader can thus better evaluate the advantages and disadvantages provided by each approach. As a major contribution, the paper shows that in the worst case the drop can be significant, while it can be minimized (without introducing any significant extra cost in terms of test generation and duration) through the adoption of a suitable observation mechanism, e.g., using Performance Counters possibly existing in the system. Suitable techniques to implement fault simulation campaigns to assess the effectiveness of different observation mechanisms are also described.

1. Introduction

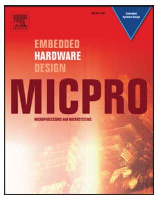
In several domains (e.g., automotive, biomedical, space and aircraft industries) electronic systems are commonly used in mission- and safety-critical applications. In these domains, misbehavior due to a defect affecting the hardware may have catastrophic effects, including hurting humans and provoking huge economic losses. Hence, there is a strong push to devise techniques able to minimize the probability that a misbehavior caused by a defect arises, and to suitably handle it in case it manifests itself anyway. When considering the latter point, different solutions have been proposed, and the best solution depends on the specific constraints of each scenario. Standards and regulations (e.g., IEC 61508 for generic safety-related industrial systems, ISO 26262 for automotive applications, RTCA/DO-254 for avionics) also play a significant role, forcing companies to devise and adopt solutions able to achieve some predefined target in terms of dependability.

Most of the electronic systems involved in safety-critical applications include a microprocessor or microcontroller. For these systems, it is possible to force programmable units to run test programs able to reveal the presence of defects by activating them and propagating their effects up to an observable location (e.g., a special memory area). Eventually, the application may trigger suitable actions to prevent catastrophic consequences, such as turning the system to a safe status, or reconfiguring it so that the faulty module is not used any more. To minimize the impact on the system, these test programs are often limited to use the time periods left idle by the core applications, or run during the start-up/power-off phases. Such an approach is referred to as *Software-Based Self-Test* (SBST) [1], and generically labeled as "functional" as it relies directly on the normal functions of the system. SBST does not require any specific Design-for-Testability (DfT) structure, although it may exploit available hardware features, and can be used to test any processor-based system, no matter whether it is a System-on-

* Corresponding author at: IIE - Facultad de Ingenieria, Herrera y Reissig 565, Montevideo 11300, Uruguay.

E-mail addresses: julio@fing.edu.uy (J. Perez Acle), riccardo.cantoro@polito.it (R. Cantoro), ernesto.sanchez@polito.it (E. Sanchez), matteo.sonzareorda@polito.it (M. Sonza Reorda), giovanni.squillero@polito.it (G. Squillero).

http://dx.doi.org/10.1016/j.micpro.2016.09.002
0141-9331/© 2016 Elsevier B.V. All rights reserved.



Chip (SoC) or a board. As a major advantage, test based on SBST can be run at the processor operational speed, thus allowing the detection of defects which are only activated at the maximum frequency. For this reason, it is often used during the manufacturing test phase as a supplement to other techniques to increase the final defect coverage.

SBST is currently adopted in quite different test scenarios, including both end-of-manufacturing test and in-field test. When applied for end-of-manufacturing test, either the automatic test equipment (ATE) drives the processor inputs while it executes the program and observes the outputs, or it loads a program into the cache of the processor, forces it to execute at full speed, and eventually extracts some test syndrome from a special (hidden) register. On the other side, when in-field SBST is considered, a common solution lies in storing the test program in a flash memory, activating its execution when required, and finally checking the content of some selected memory variables, where the test program stores its results.

When comparing the SBST solutions adopted for end-of-manufacturing test with those for in-field test, a major difference is that the former can, in some cases, benefit from full accessibility to the input and output signals of each device (such a test scenario is called "open loop test" in [32]). On the contrary, in solutions oriented to in-field testing, the tester cannot be used and existing DfT structures are in most of the cases not available (e.g., because they have been destroyed or made inaccessible to better protect the system security, or because they are not documented by the device providers). Hence, the only feasible solution for the system company in charge of developing the in-field test is to adopt a purely functional approach, i.e., without resorting to any DfT feature. Additionally, in-field constraints may be quite severe: for example, the memory area usable by the test could be limited to a specific size and location, and some faults may become functionally untestable [9] (i.e., no test stimuli exist for them under the in-field test scenario). Although untestable faults by definition cannot affect the system behavior, they may significantly limit the fault coverage that can be achieved, even using a high-quality test program. Hence, it is desirable to be able to identify untestable faults.

Concerning observability, some solutions adopted for end-of-manufacturing test may allow the continuous monitoring of all the output signals of the device under test by the ATE. On the contrary, with in-field SBST the ATE cannot be used, and thus the effects of faults are typically observed by checking, at the end of the test program execution, the values left by the program in some specified memory locations. This limited observability may significantly reduce the achievable fault coverage; some specific fault categories are known to be untestable if fault detection is only based on looking at the final memory content. In particular, faults that only affect the time behavior of the processor (e.g., by delaying some operation) found in modules such as Cache Controllers [8] and Branch Prediction Units [7] cannot be detected in this way. The test of these *performance faults* [3] can be successfully faced by resorting to the so-called performance counters existing in most of the current microprocessors and microcontrollers [4]. Alternatively, one can resort to special hardware modules that can be added to a processor, able to monitor the bus during the execution of a test program and then compute a signature. As a result, re-using any test programs developed for high-observability end-of-manufacturing SBST for in-field SBST may be either very expensive, or result in a significant drop of the achieved fault coverage. Recently, some papers specifically focused on the generation of test programs for in-field SBST [16].

The main purpose of this paper is to survey the different solutions that can be adopted in practice to support the observation of fault effects when SBST is adopted for in-field test, discussing the advantages and limitations of each of them. Secondly, the paper uses two test cases to quantitatively evaluate the benefits and cost of each observability solution: one targets the branch prediction unit (BPU) in a MIPS-like processor based system, and the other targets the cache controller logic in a dual-core LEON3 system. This paper is the first to report extensive experimental results to compare the fault coverage that can be achieved with the different solutions, thus allowing the reader to have a better understanding of the advantages and disadvantages provided by the different solutions.[1] Finally, the paper outlines some techniques to compute fault coverage figures related to the usage of an SBST approach with different observation mechanisms.

The paper is organized as follows: Section II provides an overview of the state-of-the-art in the area of SBST, with special emphasis on in-field SBST. Section III describes the different observability solutions we considered in this paper. Section IV describes the experiments we performed on the two sample systems to quantify the effects stemming from the adoption of the different observability solutions. Finally, Section V draws some conclusions.

2. Background on software-based self-test

The term *Software-Based Self-Test* (SBST) was first proposed by Chen and Dey in [14], but the approach itself has been proposed few years before under the name "*Native Mode Functional Test*" in [32] and [33]. SBST broadly identifies all test methodologies based on forcing a microprocessor/microcontroller to execute a program and checking the results to detect the presence of possible defects affecting the hardware. Indeed, the pioneering idea of testing a microprocessor with a program dates back to 1980. In [10], Thatte and Abraham devised fault models and procedures for building test programs able to detect permanent defects in different functional units of a simple processor. A wide adoption of their methodology was hindered by the difficulties in automating the generation of such test programs, especially when targeting complex processors.

In general, the usage of SBST requires:

1. Generating a suitable test program. This is typically a hard job, which is still mainly performed by hand. Moreover, the complexity and effectiveness of this task depends on the adopted metric, which in turns depends on the available information: in some cases, both RTL and gate-level models of the target system are available, while in others functional information is available, only. For the purpose of this paper, we assume that the gate-level netlist is available, and it is possible to compute the fault coverage achieved by the generated test program with respect to the most common structural fault models (e.g., stuck-at).
2. Creating an environment to support its execution. Once the test program is available, it must be stored in some memory accessible by the processor, the processor must be triggered to execute it at the due time, and the results produced by the processor during its execution must be observed. In this paper we specifically focus on the last issue.

Nowadays, the complexity of processors has significantly increased; the micro-architectural details play a fundamental role, and devices cannot be accurately modeled using information about the Instruction Set Architecture (ISA) alone. However, SBST is getting more and more important: it commonly supplements other kinds of tests, as functional programs may detect unmodeled defects that escape traditional structural tests (the so-called "collateral coverage" [11]). By definition, the functional approach tests the

[1] A preliminary version of this work was presented in [31]. In the current version we significantly extended (among the other things) the Experimental Results section by adding a further test case and improving the results analysis.