Increasing Fault Coverage during Functional Test in the Operational Phase

M. De Carvalho, P. Bernardi, E. Sanchez,
M. Sonza Reorda
DAUIN – Politecnico di Torino
Torino (TO), Italy

O. Ballan
STMicroelectronics
Agrate Brianza (MI), Italy

***Abstract*— A key issue in many safety-critical applications is the test of the ICs to be performed during the operational phase: regulations and standards often explicitly describe fault coverage figures to be achieved. Functional test (i.e., a test exploiting only functional inputs and outputs, without resorting to any Design for Testability) is often the only viable solution, unless a strict cooperation exists between the system company and the device provider. However, purely functional test often shows several limitations due to the limited accessibility that it can gain on some input/output signals. This paper proposes a hybrid approach, in which a suitable hardware module is added outside a microcontroller to increase its functional testability during the operational phase. Experimental results gathered on a couple of cases-of-study are reported, showing the feasibility of the method.**

***Keywords*— Functional test, on-line test, SBST, automotive systems, safety-critical systems**

I. Introduction

Processor-based systems are increasingly used even in application areas where safety is crucial, such as automotive and aerospace. These systems become more and more complex, and every year technology scales, bringing even smaller components. This increases circuit sensitiveness and jeopardizes safety-critical applications, which often rely on testing during the operational phase to detect possible faults and anticipate their negative effects. The growing importance of aging effects further pushes the need for effective on-line test techniques.

Standards and regulations now define constraints aimed at better guaranteeing that a sufficient level of safety is achieved in several domains. As an example, the ISO 26262 standard for automotive systems mandates the application of a continuous sequence of tests on any safety-related system on a car (e.g., in the braking system, or in the airbag control), so that failures do not cause catastrophic events during the operational life, especially due to aging effects when reaching the end-of-life wear out.

In order to match the above strict requirements different solutions can be used. Some of them are based on introducing some hardware redundancy, such as Triple Modular Redundancy, Duplication with Comparison (including Lock-step solutions), and monitoring with watchdog modules: these techniques effectively face not only permanent, but also temporary faults.

On the other side, software solutions based on introducing redundancy in the application or system code may also allow detecting faults. Both in the case of hardware and software solutions, key evaluation parameters are the achieved fault detection capability and the cost (in terms of additional hardware, additional code, decrease in performance, additional design cost, etc.).

When cost is a major concern, alternative solutions are considered, based on performing a test specifically addressing permanent faults. This test can exploit Design for Testability solutions, such as BIST, or being based on a functional approach. In the case of processor-based systems, this approach (often denoted as *Software-Based Self-Test*, or SBST [1]) forces the processor to execute a suitable program, and then looks at the produced results (e.g., in memory). The SBST approach provides some advantages, especially in terms of flexibility (the test program can be easily changed), defect coverage (the system is tested exactly in the same configuration and at the same frequency used in the operational phase) and cost (no changes in the hardware are required). On the other side, SBST adoption may be limited by the effort for developing the test program, especially when the test is developed by the system company, which often does not have access to detailed information about the device architecture.

When the SBST approach is adopted for the test during the operational phase, additional requirements may exist, in particular in terms of test duration and invasiveness [2][3][4]. The SBST approach can be extended to the test of communication [5] and system [6] peripherals.

Recently [7], it was observed that some faults may result to be untestable during the operational phase, for example because the circuitry they belong to was introduced for software debug or end-of-production test purposes, and it is not accessible any more in the operational phase, or because some signals cannot be activated in that phase (e.g., faults related to the reset signal). These faults (called *On-line Functionally Untestable Faults*, or *OLFUF*) do not affect the device behavior in the operational phase, and hence should not be the subject of any detection effort; hence, their identification may prove to be particularly importantIn this paper we focus on microcontrollers, and consider a further group of faults (that we define as *Hardly Functionally Testable*, or HFT, faults) that are hard to test during the operational phase when the functional approach is adopted. The difficulty in testing them mainly stems from the fact that they require some events on an input signal that can hardly be triggered during a functional test. However, HFT faults are not untestable (hence, they do not belong to the OLFUF set), and their occurrence could severely limit the functionality of the system, despite their limited number. Examples of HFT faults are those in the logic managing the Non-Maskable Interrupt signal (NMI), or those in the circuitry for detecting errors in the bits received through a serial channel.

In order to make the test of HFT faults possible, this paper proposes the usage of a small hardware module to be added outside the microcontroller (hence, without changing anything inside); the module is connected to the bus as a peripheral component, and can be programmed in such a way that it can trigger some events on specific target input signals. In this way the test of the HFT faults become feasible even in a functional manner.

In order to experimentally evaluate the proposed method, we considered two cases of study: the first is the circuitry driven by the NMI signal in an industrial microcontroller produced by STMicroelectronics; the second is the circuitry in charge of detecting possible errors in the bits received through an asynchronous serial channel in an open-domain UART module. In both cases we developed the proposed hardware module and wrote the test program using it to test the target HFT faults. We could thus evaluate both the advantages of the method (in terms of further faults that can be detected) and its cost (in terms of size of the hardware module).

Although the method was evaluated on a few cases of study, it is general enough to be easily adopted for the test of HFT faults in several different blocks that can be found in today microcontrollers.

The rest of the paper is organized as follows. In section 2 we better state and discuss the problem; in section 3 we describe the proposed methodology in details; in section 4 we present a couple of cases of study and report some results. Finally, some conclusions are drawn in section 5.

II. ADDRESSED ISSUE

In a typical microcontroller or (more in general) in a processor-based system it is possible to find blocks of logic that can hardly be tested during the operational phase using a functional approach.

In fact, the functional approach is performed without switching the system to any special Test Mode, nor resorting to any Design for Testability feature: the test is performed by running some special test program, and looking to the results it produces. Moreover, when functional test is used during the operational phase, the duration of the test is often a strong constraint, as well as its invasiveness (i.e., impact on the system resources): hence, the test is supposed to minimally perturb the system status.

As a consequence, it is sometimes difficult to activate during the test some external signals that are required to excite some specific piece of logic. A typical example of this situation is the logic driven by the Non-Maskable Interrupt signal (or NMI): this signal and the related logic are often very important for the system behavior (e.g., because they allow the management of critical situations, such as those related to power-down cases), but it is practically impossible to activate this signal by just working on the software side (as functional test does).

More formally, we could define the Hardly Functionally Testable (HFT) faults as those faults that require activating an input signal that cannot be activated resorting to the system software.

It is important to underline that

- HTF faults do not belong to the category of OLFUFs, as defined in [7], since the input signals they depend on are not physically tied to any fixed value, but are simply not controllable via software
- Although HTF faults are in general not so numerous, their effects may be particularly critical, and hence the importance of devising a solution for their test.

III. PROPOSED APPROACH

In this paper we propose to stimulate specific logic blocks within a microcontroller which are driven by external pins in order to increase the functional fault coverage that can be achieved during on-line testing. For this purpose we propose the introduction of a hardware module that can be programmed by an SBST program and can properly stimulate the external signals, thus suitably triggering the internal logic they drive.

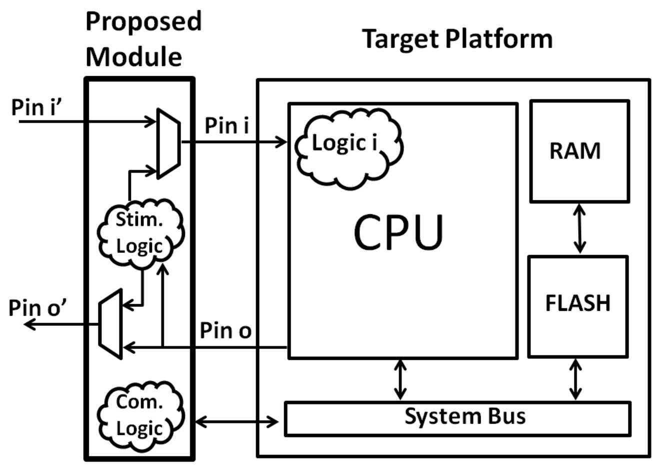


Figure 1: Proposed Framework

The scenario in which the proposed generic hardware module is used is shown in figure 1.

The module is placed outside the microcontroller device and connected to the system bus for receiving commands and retrieving data from/to the SBST program running in the CPU core. Since it is attached to the bus (typically acting as a memory mapped peripheral), it is easy for the SBST software to program the module. In addition, the microcontroller external input pins that have to be activated are driven by the output pins of the proposed module. So the proposed module is able to properly stimulate the processor/controller inputs.

The required test stimuli depend on the function performed by the pin driving the target logic block. In general, the situations supported by the proposed module are the following

- The pin must be activated (driving it from 0 to 1, or vice versa) at a given specific time
- The pin must hold the same value of a given output pin driven by the processor (denoted as pin *i* in figure 1) apart from a given clock period, in which its value is complemented with respect to the value of pin *i*.

In both cases, the logic in the CPU controlling these pins will be activated, allowing testing the target logic block (referenced to as *logic i* in figure 1).

In order to provide the proposed module with a common timing with respect to the CPU, we assume that it shares with it a clock signal.

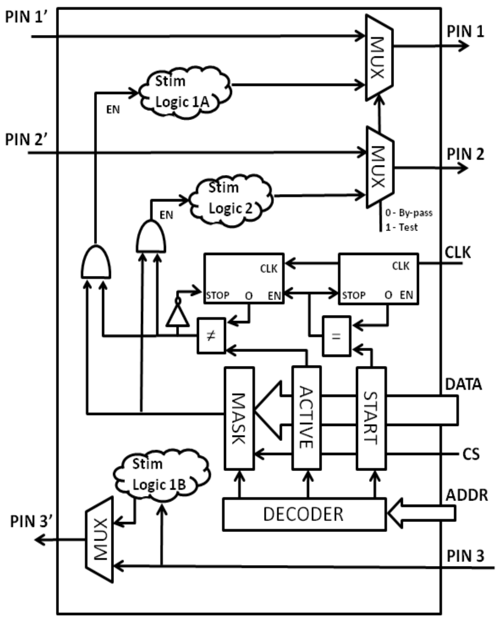


Figure 2: Proposed architecture

The proposed hardware is a generic module and its basic architecture is shown in figure 2. It has 3 registers that are used for storing commands coming from the SBST program during the programming phase: START, ACTIVE and MASK. The module has two counters (Cnt1 and Cnt2); the former enables the logic that stimulates the pin after the number of CPU clock cycles has passed given by the START register. The latter tells for how many cycles (given by the ACTIVE register) this logic should be active. The module also includes a MASK register which allows stimulating from one to many pins at the same time or even not propagating the test values to the real outputs(e.g., pin *o'* in figure 1). Since the proposed component is memory mapped, it is sufficient to write values at the address associated to each register. When START, ACTIVE and MASK registers have been properly uploaded with suitable values the module starts the test counting the number of clock cycles given by START, then activates the stimulating logic for a number of clock cycles given by ACTIVE. The MASK value masks the outputs not to be stimulated. The peripheral may have several inputs/outputs to control and can be parameterized, thus the MASK value is a binary code identifying which of them will be stimulated or not.

The proposed module has two operational modes:

- By-pass mode
- Test mode.

In by-pass mode the module is completely transparent, and does not interfere in the system operational mode.

In test mode the module controls the microcontroller target input pins and applies to them the suitable test patterns. In order to operate in test mode, the module receives commands from the SBST code (*programming phase*). After all the necessary data have been written into all registers the test starts immediately, and the target input pins are suitably stimulated (*stimulating phase*). Then, to verify that the logic has been properly tested, several status registers which have been modified during stimulation allow determining whether the logic passed the test or not. This part included in the SBST is called *check phase* and it was proposed in [9]. When this test finishes, the registers are reset and the module is switched back to by-pass mode.

It is important to highlight that only the proposed module owns two operational modes (test and by-pass) while the rest of the system always works in the normal operational mode, as mandated by the functional test paradigm.

In order to test *logic i*, it is necessary to correctly write the SBST program in such a way that the programming and stimulating phases are organized and synchronized in the proper manner.

Assuming that the functional test is performed by exploiting some idle slots left by the system application, or by temporarily suspending it, the steps to be performed by the system are:

1) The system executes the system application
2) The system switches to functional test
3) The SBST program writes values to the START, ACTIVE and MASK registers of the proposed module (programming phase)
4) The module switches to TEST mode
5) The module applies stimuli to the target pin(s) specified by the MASK register, respecting the time to start and time to remain active (stimulating phase)
6) The target logic block(s) is (are) properly stimulated
7) The SBST then compares obtained results with expected ones (check)

8) The module returns to by-pass mode and the SBST code finishes, returning the system to the system application.

Clearly, the proposed approach has some drawbacks in terms of additional circuitry to be inserted in the system, which may also increase the logic on the critical path, thus potentially impacting on the performance under the normal operational mode.

IV. Cases of Study and Results

In order to verify the feasibility of the proposed approach we considered two cases of study. The first case targets the logic driven by the NMI pin in an industrial PowerArchitecture-based microcontroller manufactured by STMicroelectronics. The second case targets the logic related to error detection in a miniUART of a 68HC11-based microcontroller. Both case-of-study systems have been synthesized under 65nm industrial technology. The synthesis was made with Synopsys Design Compiler tool. Stuck-at fault coverage results have been gathered resorting to the TMAX tool by Synopsys.

The proposed module was coded in the Verilog hardware description language and synthesized with the same industrial technology of the study cases.

A. *NMI case of study*

The first case of study is an industrial SoC platform containing a PowerPC-based CPU. The whole platform is provided by STMicroelectronics and the circuit was synthesized using a 65nm industrial technology. The platform general architecture is drawn in figure 3.

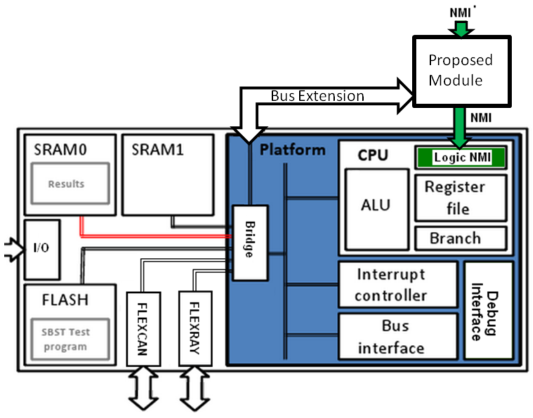
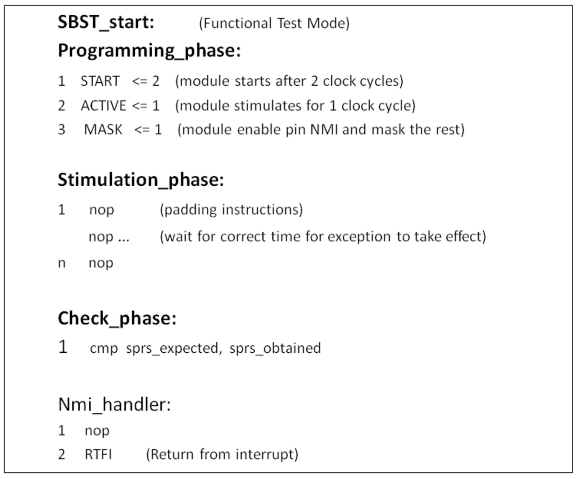


Figure 3: SoC test case architecture: the target zone where HFT faults are located is highlighted in green.

In this case of study, we focus on the logic block managing the NMI pin. In order to test such a block, the NMI signal must undergo a 1 to 0 transition which generates an exception inside the CPU core and stimulates the NMI interrupt logic (the green part shown in figure 3). Clearly, there is no way for activating such a signal in a purely functional manner. For the purpose of this case of study, the proposed module is simply in charge of activating the NMI signal based on the commands it received during the programming phase.

The SBST program developed to stimulate and observe the functionally untestable faults attached to the NMI pin is highlighted in figure 4.


```
SBST_start:        (Functional Test Mode)
Programming_phase:
1  START  <= 2   (module starts after 2 clock cycles)
2  ACTIVE <= 1   (module stimulates for 1 clock cycle)
3  MASK   <= 1   (module enable pin NMI and mask the rest)

Stimulation_phase:
1   nop       (padding instructions)
    nop ...   (wait for correct time for exception to take effect)
n   nop

Check_phase:
1   cmp  sprs_expected, sprs_obtained

Nmi_handler:
1   nop
2   RTFI     (Return from interrupt)
```

Figure 4: SBST program to test the NMI logic

The SBST program starts by configuring the proposed module, which means defining when to activate the NMI pin, how long the pin must be active and which pin to stimulate (only NMI). Some padding instructions (NOP) are necessary to make the CPU wait for the correct time to start computing the signature, and thus making the logic controlling the NMI pin observable.

Hence, resorting to the proposed module an SBST program can stimulate the NMI-related logic, which represents a small but highly critical amount of logic. This small quantity of faults which can be covered by the SBST program in this way is very important to keep product robustness and provide a safe mission. The SBST program is small, containing 3 load/store instructions, and a few padding instructions necessary to force the CPU to wait for the correct time to start observing the correct functioning of the interrupt logic triggered by the NMI pin. The SBST duration lasts around 35 clock cycles: 10 clock cycles due to module configuration, 5 padding instructions (which can be variable), and around 20 clock cycles to check whether obtained values for the special purpose registers (sprs) correspond with the expected ones, thus verifying the correctness of the logic.

The proposed external module used in this study case has 8 ports, 210 nets and 265 cells. Its adoption allows detecting 291 further stuck-at faults in the NMI logic.

The SBST program using the proposed module includes 3 instructions to program the module, plus a few instructions checking that the corresponding exception procedure has been correctly activated.

B. *MiniUART case of study*

The second case of study is an academic one in which the goal is to stimulate/test some HFT faults in the miniUART peripheral of a 68HC11-based microcontroller. The miniUART is freely available for downloading at the opencores website [8].

A common approach to test communication peripherals adopts the *loop-back* configuration [5]: during the test, two channels are selected and directly connected. One reads the data sent by the other, and by checking whether the received data correspond to the sent one, a reasonable test of the peripheral can be performed.

A major limitation of this approach lies in the fact that it does not allow to introduce errors on the channel, and hence to test the logic blocks in charge of detecting them.

In this case the proposed module is connected to the UART channel and configured in such a way to perform a loop-back communication modifying some data in order to verify that the logic in charge of protocol verification correctly works. For this particular case, the proposed module receives the data bit flow from the transmitter and can flip a given bit in the flow before providing it to the receiver. This proposed module includes some simple by-pass logic (multiplexer) in the critical path, hence slightly penalizing the performance. However, this trade-off is important for reliability sake, and satisfying standard requirements.

The NMI test is executed when the CPU has entered the test mode, thus leaving resources available for the correct execution. In this case the busses and resources should be available to perform the test. However, if real external events important for mission safety are triggered, then the SBST should go back to functional mode, and afterwards, repeat the test in another time frame.

The miniUART test framework is shown in figure 5. The logic block to be tested is the one connected to the RXD pin in the UART receiver part. In this case the proposed module is first of all in charge of flipping for one clock cycle the flow of bits coming from TXD and going to RXD. The information about when and for how long the bit should be flipped is provided during the programming phase. Using this scheme we can test the logic blocks in charge of detecting both the parity and the frame errors.

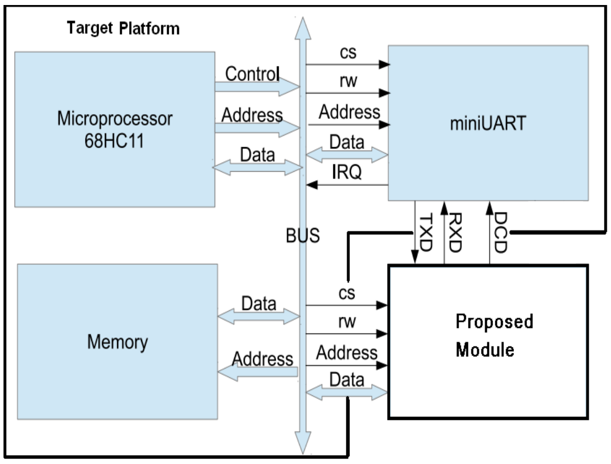


Figure 5: Proposed miniUART test framework

Secondly, the proposed module is in charge of activating the DCD signal, thus allowing stimulating the corresponding logic in the receiver. Once more, the information about when to activate this signal and for how long is given in the programming phase.

```
SBST_start:      (Functional Test Mode)
MiniUART_conf:
1  CONTROL <= (baud, data_conf, interrupt_conf, interrupt_enable)
              (Several configurations used)
Programming_phase:
1  START  <= 5  (module starts after 5 clock cycles)
2  ACTIVE <= 2  (module stimulates for 2 clock cycles)
3  MASK   <= 1  (module enable pin 1 and mask the rest)
4  TX <=  DATA  (Data transmission)

Stimulation_phase:
1  nop      (padding instructions)
   nop ...  (wait for correct time to capture stimuli)
n  nop

Check_phase:
1  cmp  Control_expected, Control_obtained
2  cmp  Status_expected, Status_obtained
3  cmp  Rx_expected, Rx_obtained
```

Figure 6: SBST program structure

Figure 6 describes the SBST program for the MiniUART case of study. It includes a prologue of 2 instructions necessary to configure the UART peripheral in terms of how the serial communication should work. Then, a second set of instructions is necessary to program the proposed module. In this case, it is sufficient to write values to the memory mapped address defined for each single register of the proposed module (START, ACTIVE and MASK). The module starts working as soon as all registers have been modified. Then, a transmission payload is written to TX (shown in red in fig 6), to make a transmission using the miniUART module. Finally, no-operation instructions (NOP) are padded in the code to force the CPU to wait for the correct time to observe the test outputs. The last part is to compare the registers with their expected values. Some methods compute the registers digest called signature, then it is compared with the expected value as proposed in [9], hence being able to tell whether the target logic passed or not the functional test. In this study case, we check the values obtained in the STATUS, CONTROL, and RX registers of the miniUART with the expected ones to detect possible faults.

Using this scheme, we developed altogether 23 SBST programs that can be run independently and able to test the UART protocol with different configurations. The forced errors in the protocol logic targeting HFT faults were:

1) *Overrun error*
2) *Framming error*
3) *Parity error*
4) *Break condition*.

The average code size for each program is 10 instructions and its duration is from 30 to 60 clock cycles, which depends on the exact time to trigger a bit flip on the protocol, thus forcing communication errors.

The proposed module implemented for this case of study contains 28 ports, 308 nets and 291 cells. Its usage allows increasing the number of detected stuck-at faults by 227: despite their limited number, these faults are particularly critical, since they impact on the ability of the module to correctly manage possible errors on the channel.

V. CONCLUSIONS

In this paper we first introduced the concept of Hardly Testable Functional faults, i.e., faults that may impact the behavior of a system, but can hardly be tested using a functional approach during the operational phase. In fact, these faults require some events on a given input signal that cannot be controlled via software.

In order to allow the test of these faults, we propose the insertion of a hardware module able to apply such stimulus in a programmable manner. Since it is memory mapped, the module is able to receive commands from a SBST program when on-line testing is performed, such that the module is only active when required to. During the mission mode it is not active and it is transparent to the system, not affecting the normal operations. The proposed module is external to the Unit Under Test, thus minimizing the invasiveness of the method.

Results gathered on a couple of cases of study show the feasibility of the approach and demonstrate that it can cover some faults that would remain undetected otherwise. Moreover, the reported results show that the cost of the proposed module is rather limited.

Currently, we are evaluating the effectiveness of our method by extending it to the test of other similar situations. We are also devising a new version of the proposed module which can be used in a wider range of cases.

VI. REFERENCES

[1] M. Psarakis, D. Gizopoulos, E. Sanchez and M. Sonza Reorda, “Microprocessor Software-Based Self-Testing,” IEEE Design & Test of Computers, vol. 27, no. 3, pp. 4-19, June 2010.

[2] A. Paschalis and D. Gizopoulos, “Effective software-based self-test strategies for on-line periodic testing of embedded processors”, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 24, n. 1, Jan. 2005, pp. 88–99

[3] E. Sanchez, M. Sonza Reorda, G. Squillero, G., “On the transformation of manufacturing test sets into on-line test sets for microprocessors”, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2005, pp. 494-502

[4] N. Bartzoudis, V. Tantsios, K. McDonald-Maier, “Dynamic Scheduling of Test Routines for Efficient Online Self-Testing of Embedded Microprocessors”, IEEE International On-Line Testing Symposium, 2008, pp.185-187

[5] A. Apostolakis, D. Gizopoulos, M. Psarakis, D. Ravotto, M. Sonza Reorda, “Test Program Generation for Communication Peripherals”, IEEE DESIGN & TEST OF COMPUTERS, 2009, vol. 26 n. 2, pp. 52-63

[6] M. Grosso, W.J. Perez Holguin, E. Sanchez, M. Sonza Reorda, A. Tonda, J. Velasco Medina, “Software-Based Testing for System Peripherals”, JOURNAL OF ELECTRONIC TESTING, 2012, vol. 28 n. 2, pp. 189-200

[7] P. Bernardi, E. Sanchez, M. Sonza Reorda, M. Bonazza, O. Ballan, “On-Line Functionally Untestable Faults Identification in Embedded Processor Cores”, IEEE/ACM Design, Automation & Test in Europe, 2013, pp. 1462-1467

[8] opencores.org

[9] P. Bernardi, et al., “Fault grading of Software-Based Self-Test procedures for dependable automotive applications”, IEEE Design, Automation and Test in Europe Conference and Exhibition, 2011, pp. 1-2