On-Line Functionally Untestable Fault Identification in Embedded Processor Cores

P. Bernardi, M. Bonazza, E. Sanchez, M. Sonza Reorda

Dipartimento di Automatica e Informatica
Politecnico di Torino – Torino, Italy

O. Ballan

STMicroelectronics
Agrate Brianza – Milano, Italy

ABSTRACT

Functional testing of embedded processors is a challenging task and additional constraints are imposed when a functional test procedure has to be executed on-line. In the latter case, a significant amount of the processor faults cannot be detected since related to the debug/test circuitry or because of memory configuration constraints. In this paper we identify several sources of on-line functional untestability and propose a set of techniques to exactly measure their impact on the fault coverage. Experimental results related to an industrial case study are reported, showing that the fault coverage loss due to the considered untestability sources may reach more than 13%.

1. Introduction

Nowadays, being able to functionally test embedded microprocessors is a major concern for industry. While scan-based approaches are still dominating manufacturing test, functional testing is a recognized strategy to self-test embedded processors functionalities during their mission. By functional test we mean here a technique that aims at detecting possible faults by just acting on the functional inputs and observing the functional outputs of a processor, without resorting to any Design for Testability. Since a functional test is typically based on forcing the processor to execute a suitable test program, and then on observing the results the test program produced (e.g., in terms of memory content), processor functional test is also referred to as Software-Based Self-Test, or SBST [6].

In the automotive sector, the self-test of the processor(s) included in an Electronic Control Unit (ECU) during the operational phase is mandated by the ISO 26262 standard, which indicates three levels of confidence with respect to the safety of the car depending on the coverage figure. For very critical environments, such as airbags or drive-by-wire functions, the standard mandates for 98% of fault coverage. In such a scenario, it is crucial to distinguish untestable faults, so that they can be removed from the list of faults to be tested. Since functional test is a common solution for fulfilling the on-line test requirements mandated by such regulations, better understanding and clearly identifying which faults cannot be tested following such solution is a major practical concern.

978-3-9815370-0-0/DATE13/©2013 EDAA

Several works already dealt with untestable faults, namely introducing structural and functional untestability and proposing techniques for their identification [1]; while many works focused on untestable faults in generic combinational and sequential circuits [2][3][4], only few specifically studied the sources of untestability in microprocessors [5][9]. The main issue in this case is that the identification of functionally untestable faults requires analyzing the behavior of all instructions and sequences of instructions, thus distinguishing the sequences of values that can be applied to the inputs of combinational blocks from those that cannot. In turn, a fault can be classified as functionally untestable if there is no sequence of instructions that can either excite it, or force it to produce visible effects, or both.

In this paper we introduce and formally define *on-line functionally untestable faults*; they are faults related to

- Hardware resources within the embedded processor that are no longer used along the mission behavior, having been introduced for manufacturing test and debug, only
- Address generation management, whereas the allocated memory space is smaller than the maximum allowed size, or when some memory portions cannot be accessed freely during the in-field behavior.

In particular, we focus on three sources of on-line functional untestability, related to:

- Scan circuitries
- Debug circuitries
- Addressing resources.

In this paper we also describe a methodology for practically identifying on-line functionally untestable faults: the method is mainly based on modifying the circuit for the purpose of untestable fault identification by connecting some carefully selected signals to fixed values; in this way on-line untestable faults are transformed into structurally untestable faults, and can thus be identified by the usual EDA tools. Results obtained using the proposed technique are quite significant: we observed about the 13% of coverage lost due to on-line functionally untestable faults on an industrial system-on-chip for automotive including a 32-bit embedded microcontroller.

The paper continues as following. Section 2 provides some background about the addressed issue. Section 3 describes the key elements of the proposed strategy and then describes it in details. Section 4 shows the results

obtained on an industrial case study. Section 5 draws some conclusions.

2. Background on Untestable Faults

According to the literature [1], an *untestable fault* is a fault for which no test exists.

Untestable fault identification is often performed by starting from the combinational block each fault is located in. The first operation is typically the identification of those faults that, being related to some redundant logic, cannot be tested even if we had full access to the inputs and outputs of the combinational block. These faults are normally called *structurally untestable faults*.

When dealing with processors, the possibility of testing those faults that are not structurally untestable strongly depends on the target fault model and on adopted test approach: for example, all structurally testable stuck-at faults in a combinational block can be tested when resorting to a scan solution. Conversely, only a relatively small subset of structurally testable path delay faults can be tested resorting to a functional approach [5][7][9].

When processor-based systems are used in safety-critical applications, it is often required to monitor their health by routinely performing some kind of test; while BIST solutions provide several advantages in this scenario, they require the system to be suitably designed and the BIST mechanism to be activated on field, possibly without significantly impacting on the system status. For these reasons an alternative solution is based of performing a functional test, based on forcing the processor to execute a test program and observing the produced results [8]. Specific target fault coverage figures must be achieved in both cases to fulfill safety-related constraints.

In general, faults that cannot be tested resorting to the functional approach (i.e., for which no test program exists able to detect them) can be defined as *functionally untestable*.

When following the functional approach, it is well known that a major obstacle lies in the generation of suitable test programs, able to reach the target fault coverage figure. Hence, it is crucial to be able to preliminarily identify functionally untestable faults, and prune them from the fault list, thus reducing the test program generation effort and improving the testable fault coverage figure.

The scenario can be even further detailed when the functional approach is used for on-line testing: in this case significant limitations may exist in terms of accessible input and output signals. As an example, it may be impossible with a purely functional test program to activate the reset signal to the processor. Similarly, the output behavior of the processor cannot be fully observed in this scenario: as an example, a fault only producing a delay of some clock cycles in a specific memory access cannot be detected, since only the final content of some memory variables is typically checked at the end of the test program.

As a result, we can state that when the functional approach is used for on-line test, some faults cannot be tested, even if they are functionally testable: we call these faults *on-line functionally untestable*.

3. On-line Functionally Untestable fault identification

When testing on-line a microprocessor-based system-on-chip through functional approaches, it is usual that some resources used for design and silicon debug, and for manufacturing test are no more accessible. These structures can basically be classified into two groups:

- Design for Testability circuitries controlled directly on the boundary of the chip by a tester during manufacturing test, including
  - Scan chains
  - Boundary scan and IEEE 1500 structures
  - Built-in self-test modules
- Design for Debug units controlled by external controllers during silicon and software debug, such as Nexus-compliant modules.

The access ports of these modules are often soldered to ground or Vdd, or pulled to a fixed logic value when the system is put in its mission field. This is sometimes done for security reasons, too, thus reducing the possibility to exploit these access points for supporting an illegal attack to the circuit.

The direct consequence of such a final configuration of the system is the appearance of a further set of untestable faults that we call *on-line untestable faults*. Such faults are testable until the structures they are related to are used, but not in the final environment.

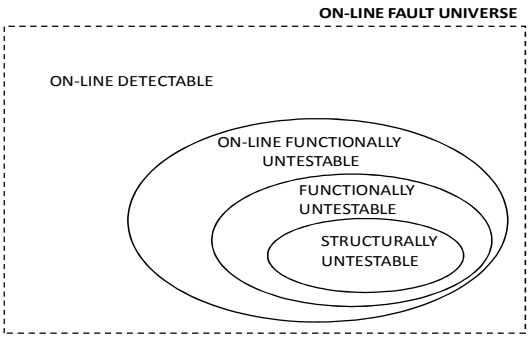

Fig. 1: On-line fault categories and their relationship.

Figure 1 exemplifies this scenario; in the on-line fault universe, the on-line functionally untestable fault set is a superset of the functionally untestable faults.

The main purpose of the paper is providing methods for the identification of such category of faults in the embedded processor. This task is not trivial, since in most cases the circuitries originating on-line untestabilities are deeply embedded in the processor functional modules. Our

identification process exploits the capabilities of conventional tools capable of tracing scan chains and distinguishing structurally untestable faults in combinational circuits.

In the following sections, we will also discuss the functional untestabilities introduced by the mission configuration of the system; in simple words, it is usual to observe that some functionalities of the processor are not fully exercised. This is normally due to usage of generic modules in a specific configuration context, such as a memory map partially covering the potentially available memory space.

3.1 Scan circuitries

A set of on-line functionally untestable faults is related to the scan chain circuitries.

Since the scan chain is no more used along the mission behavior, most of the faults potentially affecting it may not be considered anymore.

Figure 2 shows a stuck-at mux scan flip flop: in the fault-free circuit, the serial input SI is never connected to the flip flop, which is always fed with the functional input FI. Supposing the scan enable SE signal to be forced to the logic value '1' to make the scan chain working, the stuck-at-0 fault on the SE signal is untestable. As well, stuck-at-1 and stuck-at-0 on the SI signal are untestable.

As a result of this analysis, it can be stated that the only fault that needs to be taken into consideration is the stuck-at-1 on SE; this fault can erroneously connect the SI signal to the flip flop, thus changing the mission behavior of the processor.

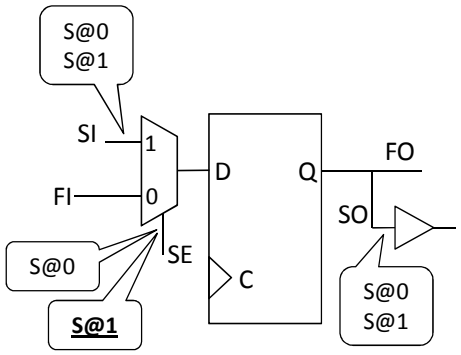

Fig. 2: mux scan structure and faults related to the scan behavior.

Automatic identification of these faults can be done quite easily by using any EDA tool able to trace the scan chain. This means for any kind of flip flop reached by the trace operation, identifying the SI and SE signals, and then removing the stuck-at-1 and stuck-at-0 faults on the SI signal and the stuck-at-X fault on the SE signal, where X is the logic value for the selection of the functional mode (i.e., stuck-at-0 in the example).

Finally, it has to be mentioned that buffers and inverters may exist on the scan path between flip flops; all faults concerning these buffers have also to be considered as on-line functionally untestable, analogously to the faults affecting SO in figure 2.

3.2 Debug circuitries

Debug circuitries are included in most of the available embedded processors. Such modules are mainly intended to observe and, eventually, manipulate registers content and to perform controlled execution (e.g., step by step, run until breakpoint, etc.) of a software application.

As an example, the Nexus port is a popular debug infrastructure, which permits to observe and control the processor status by enabling an external debugger to access its structures.

During the mission behavior, such an external debugger is not clearly connected to the system: therefore, there are circuitries no more used in the processor and generating on-line functionally untestable faults. Indeed, it is crucial to identify such faults when the objective is the achievement of the highest possible coverage.

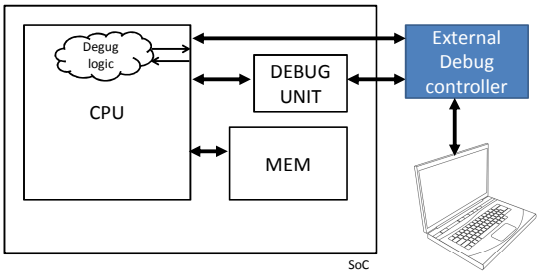

Fig. 3: Debug components in a System-on-Chip.

To the sake of clarity, we distinguish control abilities from observation capabilities related to the debug interface, and we provide a technique to quickly identify on-line functionally untestable faults for both cases.

3.2.1 Untestable faults due to unused control logic

The top level of the embedded microprocessor can be reached by signals devised to manage the processor functionalities during debug sections. Through these signals it is possible from an external equipment to monitor and even to manipulate the processor status.

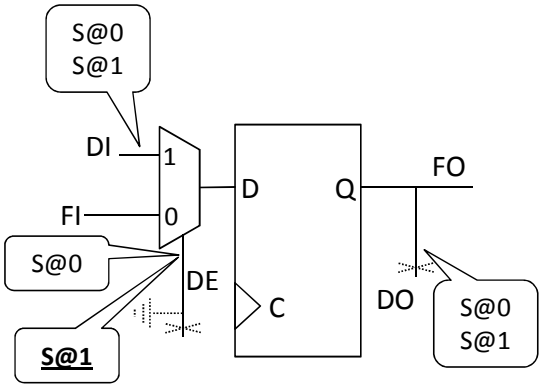

Fig. 4: debug circuit configuration for enabling flip-flop manipulation.

As an example, a generic debug structure is shown in figure 4, with a Debug Enable signal (DE), enabling a value to be forced from the outside to replace the current flip flop value. During the on-line behavior, DE is no longer used since the external debugger is no more connected. Indeed, DE generates an on-line functionally untestable stuck-at-0, as well as the stuck-at-X faults of the Debug Input (DI) signal.

To quickly identify the faults that are on-line functionally untestable, we resorted to a circuit manipulation followed by a structural untestability analysis. The procedure is composed of the following steps:

1. connect to ground or Vdd (i.e., tied'0 or tied'1) all CPU inputs related to debug and showing a constant value
2. run any EDA tool able to identify structural untestable faults
3. remove the identified faults from the fault list.

3.2.2 Untestable faults due to unused observation logic

Similarly to unused control logic, also the signals devised to transport data from the processor to the external debug equipment are possible sources for on-line functionally untestable faults.

As a matter of fact, the values on these signals are not read when in the in-field application environment. Therefore, all faults related to circuitries only devoted to carry values to the outside of the chip for debug purposes should be considered as on-line functionally untestable.

In figure 4, the signal Debug Output (DO) is left floating when the debugger is disconnected, therefore the related faults do not affect the behavior of the chip.

Again, we can resort to a circuit manipulation followed by a structural untestability analysis to identify this group of on-line functionally untestable faults:

1. Unconnect (e.g., leave floating) all CPU outputs related to debug
2. run any EDA tool able to identify structural untestable faults
3. remove the identified faults from the fault list.

3.3 Memory map

Another source of on-line untestability is related to the organization of the memory space. In fact, it is quite common to have only a part of the available addresses really accessible while a generic address management component is included in the System-on-Chip.

Let us consider the following realistic scenario. A RAM and a flash memory, whose size is 1024x8 and 4096x8 bits, respectively, are connected to a system bus with 32 bits used for addressing. In this case, the processor is potentially able to manage 2^{32} addresses, but it is required to intervene only on a small subset of addresses; if the considered RAM and flash memory core are mapped one after the other, starting from address 0, then only 12 bits of the address bus will really be used.

Unused address bits originate logic gates stuck to a solid value along all the mission behavior; therefore, they are source of on-line functional untestability; as a matter of fact, during the in-field behavior it will not be possible to change the value of every address bit by executing instructions.

The set of modules affected by this phenomenon includes address generation, prediction and virtualization units. Due to the memory address restriction, all registers used for storing addresses will always show a constant logic value ('0' in the proposed explanatory example) and logic gates used for address manipulation will be only partly used.

In an address generation unit, for instance, it means that the inputs of the adder used in a branch address calculation are fed with some constant inputs, thus compromising the final coverage.

In a prediction unit, many bits in the registers used to save a branch addresses are stuck to a value, and the same negative effect is shown by units for virtual memory management.

To quickly identify the faults that are on-line functionally untestable, we resorted again to a circuit manipulation followed by a structural untestability analysis:

4. connect to ground or Vdd (i.e., tied'0 or tied'1)
    a. input and output of those flip flops showing a constant value in any register involved in address manipulation
    b. input of the modules used to manipulate addresses (i.e., adder for branch calculation)
5. run any CAD tool able to identify structural untestable faults
6. remove the identified faults from the fault list.

Figure 5 graphically illustrates which faults will be identified as untestable in a D flip-flop with active low reset whose value is constant to '0'. This example is falling in the case 1.a.

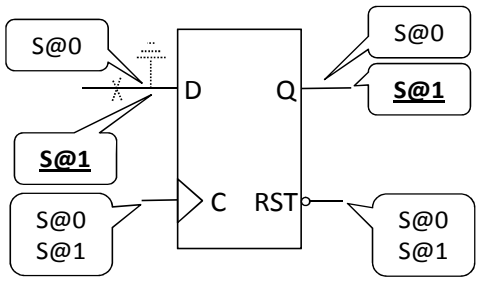

Fig. 5: On-line functionally untestable faults in a DFF with active low reset whose value is constant to '0'.

The structural analysis returns only 2 testable faults, stuck-at-1 on D and stuck-at-1 on Q. If this is a flip flop in a register storing an address for any reason, an error will propagate in the system and produce an exception.

In our solution to identify on-line untestable faults, we also connect the output of the flip flop to tied values. This action permits to propagate a tied value to the logic gates connected to the flip flop in case the used tool stops the untestable identification process at flip flops.

Figure 6 graphically illustrates the working principle of the technique. When forcing a tied value to a net, i.e., driven by a flip flop with on-line constant value, structural untestable faults are identified by just looking at the structural properties of the connected circuit portion.

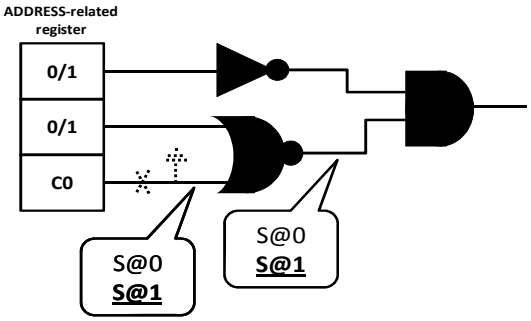

Fig. 6: On-line functionally untestable faults in a DFF with active low reset whose value is constant to '0'.

Similarly, any circuitry used for address manipulation has to be investigated; for instance, some of the inputs of the adder used for address generation may be forced to tied values, thus allowing the identification of untestable faults.

4. An industrial case study

We experimentally evaluated the proposed strategy on a commercial System-on-Chip including a processor with 32-bits address and data parallelism. This activity was performed in a larger context; in particular, it is part of the effort devised to work out a software-based self-test library with high fault coverage capabilities. In fact, since the chip is employed in the automotive field (e.g., for the airbag control), it has to cope with the ISO 26262 standard. From this point of view, it is fundamental to screen out any on-line functionally untestable faults since the coverage requirement is quite strict.

The proposed strategy was applied when the self-test suite of functional programs was already quite mature. As it will be better described later in this paragraph, the identification of on-line untestable faults permitted to raise by about 13% the stuck-at fault coverage.

We set up a flow corresponding to the procedures described in section 3, composed of several phases and adopting several commercial and ad-hoc CAD tools. To summarize and generalize, we identified three major activities able to effectively lead us to a secure and low resource consuming result:

1) Search for sources of untestability
2) Circuit manipulation (possible)
3) Screen out on-line functionally untestable faults through
    a. Direct pruning from the fault list
    b. Structural untestability checking.

The considered embedded processor is equipped with dedicated test and debug circuitries. It is fully scannable, with mux-Dff composing several scan-chains. It is connected with a Nexus-compliant module enabling general and special purpose register manipulation, as well as memory data control. Furthermore, it supports a 32 bits address bus connecting two memory cores, a 128K SRAM and a 512K Flash.

The embedded processor core totally accounts for 214,930 stuck-at faults.

The first task was dealing with on-line untestable faults due to the scan chain. In this case, we first investigated the nature of the used scan flip flops by doing some experiments using Tetramax. The main purpose of the experiments was to verify the assumptions regarding untestable faults resulting from a constant SE value; we then tied the SE signal to a fixed value and observed the result from Tetramax; on-line untestable faults are classified as "untestable due to tied value - UT" by the tool.

Then, with the list of faults to be safely removed from each scan-ff fault list, we used an ad-hoc tool able to trace the chain and directly select the on-line functionally untestable faults.

As a result we pruned about 20K stuck-at faults, corresponding to about 9% of the total of faults.

We then considered the debug circuitries. Coherently with the distinction between untestable faults due to unused control and observation logic, discussed in paragraph 3.2, we devised two strategies.

To prune faults related to unused control logic, we resorted to a preliminary analysis based on high-level code coverage metrics. It is quite usual to exploit these metrics, such as toggle, switching and condition coverage, to have an idea of which parts of the circuit are used and which are not (this is normally reflecting in a localized low coverage). Since working on a quite mature self-test program suite, any signal still showing no activity was identified as suspect. In particular, we first concentrated our investigation on the inputs of the CPU, then on its control and data-path blocks.

The result has been the selection of 17 signals, related to the debug functionalities driven by a Nexus unit collocated outside the processor core or directly by ports of the chip. They include an entire JTAG access port and several control signals used to manage the processor behavior when the chip is connected to an external host PC for debug purpose.

We proceeded to circuit manipulation by tying these signals to fixed values. The manipulated circuit was finally fed to Tetramax, which identified a significant number of Untestable Faults. In particular, we identified 4,548 on-line functionally untestable faults.

To remove faults related to unused observation logic we first looked at the boundary of the CPU for signals neither explicitly (e.g., being part of the system bus) or implicitly (e.g., provoking a system reaction) participating to the system behavior. This is the usual case for debug signals carrying out processor state information (i.e., the registers content).

Since the evaluation of the fault coverage of a self-test program procedure is obtained by only observing the system bus, we modified the circuit by disconnecting two 32-bits busses directly providing general and special purpose register values to be only captured along debug sessions.

The obtained circuit was finally analyzed with Synopsys Tetramax: the produced fault list is smaller than the one produced for the original circuit. The removed faults, 2,357 stuck-at faults, are corresponding to on-line functionally untestable faults due to reduced observability.

As a last source of investigation, we looked for the memory space organization. Since the e200z0 is implementing a 32 bits address bus while connecting two memory cores, a 512K SRAM and a 512K Flash, it has to be deduced which address bits could never move. Let us consider that:

- Flash address range is 0x0007_8000 to 0x0007_FFFF
- RAM address range is 0x4000_0000 to 0x4001_FFFF

This configuration implies that only the 18 less significant bits and the 30th bit can assume legally both '0' and '1'.

Circuit manipulation was applied for connecting to ground the unused bits of all register in the address generation unit, including those belonging to the Branch Target Buffer module.

Tetramax analysis brought out 3,610 structurally untestable faults.

Table I: summary of the identified on-line functionally untestable faults

	On-line functionally untestable faults	
	[#]	[%]
Original	0	0
Scan	19,142	8.9%
Debug	4,548+2,357	3.2%
Memory	3,610	1.7%
TOTAL	**29,657**	**13.8%**

Table I summarizes the overall result achieved. The coverage loss due to the identified sources of on-line functional untestability was 13.8% of the entire fault list. A relevant aspect that needs to be underlined is related to the time required to complete this analysis. From the human cost point of view, it required a skilled test engineer to identify untestability sources; this operation was accomplished in about 1 week some days of work. From the CPU time point of view, the modified circuit is analyzed by Tetramax in less than 1 second.

5. Conclusions

When testing on-line a processor-based system using a functional approach, techniques to identify faults for which no test program exists may allow to significantly reduce the effort for test program generation and to better evaluate the achieved fault coverage.

This paper contribution is two-fold: from one side it introduces the definition of on-line functionally untestable faults, reporting some examples of this category of faults; from the other, it proposes some techniques to make their identification feasible in industrial practice, i.e., by resorting to commercial EDA tools. Some experimental results related to a real processor core are also reported, showing both that the number of on-line functionally untestable faults is not negligible (about 13% of the total stuck-at fault list) and that their identification is practically feasible.

We are currently working to extend the proposed technique to other fault models.

6. References

[1] L.-T. Wang, C.-W.n Wu and X. Wen, VLSI Test Principles and Architectures, Elsevier, 2006

[2] J. Raik, H. Fujiwara, R. Ubar, A. Krivenko, Untestable Fault Identification in Sequential Circuits Using Model-Checking, Proc. IEEE Asian Test Symposium, 2008, pp. 21-26

[3] Syal, M.; Hsiao, M.S., New techniques for untestable fault identification in sequential circuits, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 5, no. 6, 2006, pp. 1117 – 1131

[4] H.-C. Liang; C. L. Lee; Chen, J.E., Identifying Untestable Faults in Sequential Circuits IEEE Design & Test of Computers, Vol. 12 , No. 3, 1995, pp. 14-23

[5] W.-C. Lai; Krstic, A.; Kwang-Ting Cheng, Functionally testable path delay faults on a microprocessor, IEEE Design & Test of Computers, vol. 17, no. 4, 2000, pp. 6-14

[6] Psarakis, M.; Gizopoulos, D.; Sanchez, E.; Sonza Reorda, M., Microprocessor Software-Based Self-Testing, IEEE Design & Test of Computers, vol. 27, n. 3, pp.4 - 19, May-June 2010

[7] Gurumurthy, S.; Vemu, R.; Abraham, J.A.; Saab, D.G., Automatic Generation of Instructions to Robustly Test Delay Defects in Processors, Proc. IEEE European Test Symposium, 2007, pp. 173-178

[8] Gizopoulos, D., Online Periodic Self-Test Scheduling for Real-Time Processor-Based Systems Dependability Enhancement, IEEE Transactions on Dependable and Secure Computing, vol. 6, no. 2, 2009, pp. 152-158

[9] Bernardi, P.; Grosso, M.; Sanchez, E.; Sonza Reorda, M., A Deterministic Methodology for Identifying Functionally Untestable Path-Delay Faults in Microprocessor Cores, Proc. IEEE International Workshop on Microprocessor Test and Verification, 2008, pp. 103-108