Software-Based Self-Test for Delay Faults

Michelangelo Grosso, Matteo Sonza Reorda, Salvatore Rinaudo

Software-Based Self-Test for Delay Faults

Michelangelo Grosso, Matteo Sonza Reorda, Salvatore Rinaudo

STMicroelectronics s.r.l., AMS R&D, Torino, Italy
michelangelo.grosso@st.com
Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, Italy
matteo.sonzareorda@polito.it
STMicroelectronics s.r.l. AMS R&D, Catania, Italy
salvatore.rinaudo@st.com

Abstract. Digital integrated circuits require thorough testing in order to guarantee product quality. This is usually achieved with the use of scan chains and automatically generated test patterns. However, functional approaches are often used to complement test suites. Software-Based Self-Test (SBST) can be used to increase defect coverage in microcontrollers, to replace part of the scan pattern set to reduce tester requirements, or to complement the defect coverage achieved by structural techniques when advanced semiconductor technologies introduce new defect types. Delay testing has become common practice with VLSI integration, and with the latest technologies, targeting small delay defects (SDDs) has become necessary. This chapter deals with SBST for delay faults and describes a case of study based on a peripheral module integrated in a System on Chip (SoC). A method to develop an effective functional test is first described. A comparative analysis of the delay faults detected by scan and SBST is then presented, with some discussion about the obtained results.

Keywords: Software-based self-test, transition delay faults, small delay defects, VLSI, microcontrollers, peripherals.

1 Introduction

Testing at the end of manufacturing is a mandatory requirement for digital integrated circuits, to guarantee product quality and minimize the number of field returns. Its cost constitutes a large part of the overall budget, and consequently designers and product engineers collaborate to find the best solutions in terms of test coverage and application costs for the products. The inclusion of additional Design-for-Testability (DfT) dedicated structures within the chip is considered a valid approach to simplify and accelerate test generation and application: the most common approach, in digital logic, is the use of scan chains, which provide direct controllability and observability to most flip-flops in the circuit. Today's scan chain-based methodologies overcome many limitations of the basic approach. Some examples include:

- Scan compression, to reduce the test pattern size and alleviate the memory requirement on the tester;

- On-chip clock controllers, to use available on-chip oscillator and phase-locked loop (PLL) for applying patterns at-speed, i.e., at the nominal circuit frequency;
- Power-aware pattern generation, to avoid the excessive energy dissipation during test due to a switching activity higher than normal.

Alternative and complementary approaches to scan chain-based testing have been developed and used in the past to provide a wider range of methods to designers and product engineers. Among those, Software-Based Self-Testing (SBST) methods [1] are based on the application of functional stimuli to an on-chip microprocessor, by forcing it to run a specific piece of code. With such a kind of stimulation, it is possible to guarantee the detection of structural faults within the logic, at the nominal circuit frequency (at speed) and without extra power consumption; however, test generation and coverage assessment processes are not as standardized, automated and widespread. The adoption of advanced semiconductor technologies even for safety-critical applications, requiring a high level of reliability, triggered the usage of SBST for in-field test, in the form of Self-Test Libraries (STLs) developed by the semiconductor company manufacturing the device and integrated by the system company in the application code [2].

While most of the papers describing techniques to generate SBST programs and assess their effectiveness focused on stuck-at faults, some of them also dealt with delay faults [3-8], whose importance is growing with shrinking semiconductor technologies. Several researchers (e.g., [9]) highlighted the fact that the percentage of functionally untestable delay faults (i.e., delay faults that cannot produce any failure when the circuit works in the operational mode) may be significant in many cases, thus reducing the achieved fault coverage. Clearly, the ideal approach would be to remove untestable faults from the fault list when computing the achieved fault coverage [10]. Unfortunately, given the complexity of modern processors, the task of identifying functionally untestable faults with a scalable effort still remains an open problem [9][11].

Delay defects are usually modeled as path delay faults or transition delay faults. Transition delay faults are more easily handled by Electronic Design Automation (EDA) tools and test suites targeting stuck-at and transition delay faults in combination with scan are quite common in the industry. However, with the continuous shrinking of geometries and rising of frequencies, and as a direct consequence of process variations, the impact of more subtle delay defects has been rising, and the specific case of *Small Delay Defects* (SDDs) required in the last years the development of more advanced test generation techniques [12].

In this chapter, which extends a previously published paper [13], we propose a SBST methodology for testing a digital communication peripheral embedded in a mixed-signal ASIC device at the end of manufacturing focusing on both stuck-at and transition delay faults. The manually developed test stimuli include a specific code to be run by the embedded microcontroller, in parallel with the interaction from the outside handled by an Automatic Test Equipment (ATE). The effectiveness of the approach in detecting SDDs is also evaluated and discussed.

The goal of the chapter is first to demonstrate that SBST can be used to improve or replace part of traditional test procedures for digital logic, thus improving test coverage while containing test application cost. As a matter of fact, functional/embedded software-driven testing parts are commonly employed for analog components in mixed-signal circuits, disregarding the coverage that they may inherently obtain on the digital logic. Secondly, we compared the list of faults detected by the proposed SBST technique with the faults detected using scan. Results show that due to designer choices, some faults can only be detected resorting to a functional approach, while some of the faults which are only detected by the scan test proved to be functionally untestable, and hence their detection produces some overtesting. We also highlight the specific effort and computational time required for the process of fault grading the developed procedures, depending on the chosen fault model and observation strategy. Finally, for the first time, we compare the transition delay coverage and small delay defect coverage obtained with the use of the SBST approach.

In summary, the contribution of the chapter lies from one side in proposing a technique to guide the test engineer in the generation of suitable SBST tests for a peripheral module, on the other on reporting detailed experimental results related to the stuck-at and delay (transitions and small delay) fault coverage figures achievable with SBST and scan on a real industrial case study.

This chapter is structured as follows: Section II provides an essential background to appreciate details and motivations of the work; Section III describes the flow used to develop the test set and to evaluate its test coverage. Experimental results on a case study are reported in Section IV, and conclusions are drawn in Section V.

2 Related Works

This chapter focuses on the end-of-manufacturing test of a case study corresponding to a peripheral module managing communications with the outside of a System on Chip (SoC).

Test development for digital circuits relies on the definition of *fault models*, abstract models that represent the behavior of the circuit in the presence of a manufacturing defect. This kind of modeling provides a mathematical method to analyze and measure the effectiveness of a test in detecting the defect effects, and hence in discriminating good and faulty devices, using a logic netlist representation of the circuit. The fault coverage of a set of stimuli is represented by the ratio of faults that cause a difference between the real circuit outputs and the expected ones, and the total number of faults.

The most common and widely used fault model for digital blocks of logic is the stuck-at, corresponding to each single node in the circuit being fixed at the 0 (stuck-at-0) or 1 (stuck-at-1) logic level. The number of stuck-at faults is simply the number of nodes multiplied by two. To complement the effectiveness of a test targeting stuck-at faults, other fault models are used that consider other defect effects, such as bridging and delay faults.

Delay faults represent the behavior of a block of logic that is slower than expected, due, e.g., to the increased resistivity or capacitance of a circuit structure. Delay defects are usually modeled as path delay faults or transition delay faults. The former model is defined as the cumulative delay of a combinational path that exceeds some specified duration (e.g., the maximum propagation delay). The latter consists in a larger than normal delay in the toggling of the logic value of a node (slow-to-rise or slow-to-fall); such additional delay is considered to be large enough to cause an error on an output or on the next clock front, when data sampling occurs. The transition delay fault model is widely used due to its inherent simplicity, as it does not require timing models for pattern generation, and Automatic Test Pattern Generation (ATPG) algorithms are based on the ones for stuck-at faults.

Experimental data [14-15] shows that the smaller delay defects are more likely to happen in the circuits. However, the transition delay fault coverage does not guarantee the detection of subtle defects that cause small delays within a combinational path. In fact, ATPG tools usually aim at covering the transition delay fault on a specific node with the lowest effort, i.e., not necessarily activating the worst path passing through that node. The detection of Small Delay Defects (SDDs) requires more specific patterns, and therefore more sophisticated ATPG and fault simulation flows [12][16].

With respect to traditional transition delay fault ATPG flows, this kind of analysis requires timing analysis data – usually expressed in the Standard Delay Format (SDF). Fig. 1.*a* (adapted from [16]) shows how a specific transition delay fault can be activated through different paths. Considering the endpoint flip flop always observable and the launch and capture clocks perfectly balanced, the delay fault on the combinational logic will be detected by the test activating the transition only if the additional delay is larger than the path slack (Fig. 1.*b*). Therefore, to detect a smaller delay defect, the transition has to be activated on a path with smaller slack, and thus this information has to be computed by the ATPG tool. This translates into additional algorithm complexity.

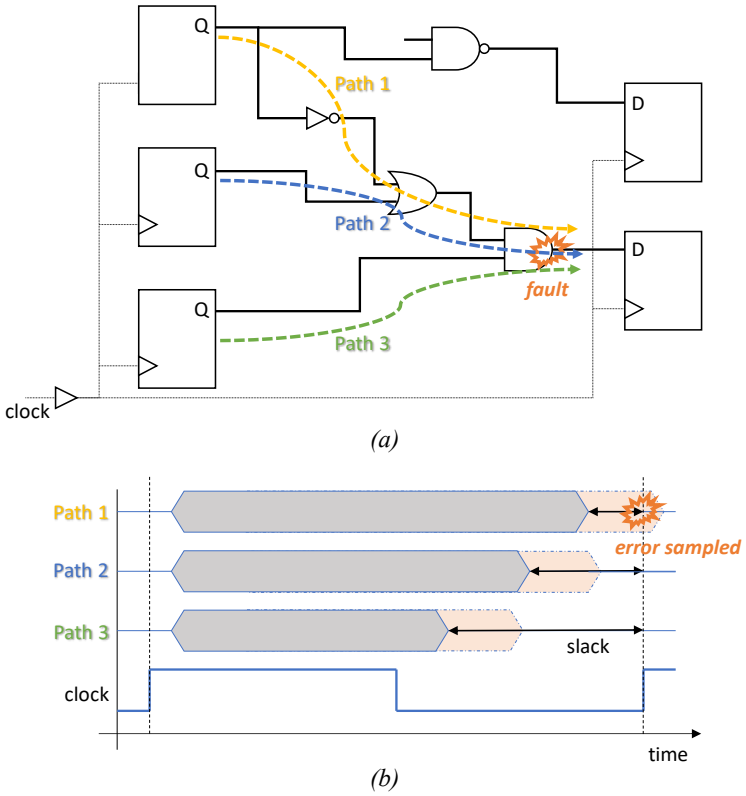

Fig. 1. In *a*) three different logic paths through which a fault can be activated in a sample circuit. In *b*) the timing diagram of the three paths showing the slack in each case: a small additional delay (highlighted in orange) causes an observable effect only along Path 1.

The size of the delay to be considered is strictly correlated with the implementation technology and on slack distribution in the circuit. ATPGs usually address faults within a maximum timing margin (*max_tmgn*), representing the slack limit for targeting faults at their minimum slacks. To provide more expressive test quality measures with respect to fault coverage, different metrics are described in the literature, such as Delay Effectiveness (DE, [16-17], which takes into account the size of the delay fault and the minimum slack of a path that can activate it, and probabilistic models considering slack and fault size distributions (e.g., [19-19]). DE is defined as the ratio between the integral of the cumulative distribution of detected fault slacks F_D and the integral of the cumulative distribution of total fault slacks F_T within *max_tmgn*:

$$DE = \frac{\int F_D(t)dt}{\int F_T(t)dt} \qquad (1)$$

Test of digital modules is typically performed resorting to DfT techniques, such as scan. While it guarantees an easy to apply and effective solution for stuck-at faults,

scan is known to have some criticalities when delay faults are considered. In such a case, Launch on Capture (LoC) and Launch on Shift (LoS) can be used [20], which are widely supported by commercial tools. Both LoC and LoS are known to produce some overtesting, since they perform the test with full freedom in controlling and observing the flip-flop state. In normal operational conditions this is clearly not the case. When considering path delay faults, the overtesting issue can be tamed by identifying functionally untestable paths and removing them from the target fault list [21-22]. While exact solutions are hardly scalable, approximate ones have also been proposed [10]. The role of temperature when facing delay faults has been explored in [23].

As an alternative to DfT solutions, functional ones (based on stimulating the circuit acting on the functional inputs and observing the functional outputs, only, without any DfT support) provide the advantage of not requiring any hardware overhead nor producing any overtesting. On the other side, test stimuli generation is not automated in this case, and its cost is clearly much higher. This solution may be particularly attractive for SoCs including at least one processor, where a functional test takes the form of a program suitably written to excite the target faults and make their possible presence visible on the circuit outputs (Software-based Self-Test, or SBST) [1]. Previous papers explored techniques to guide the test engineer in the development of suitable SBST programs targeting stuck-at and delay faults [4-6][24]. Others focused on a comparison between the Fault Coverage achievable with scan with respect to the one of SBST, taking also into account the untestable delay faults [9]. Some works also tried to provide techniques allowing to automate the generation of such programs [3][7], possibly resorting to a hybridization between DfT and SBST [25]. Finally, some recent works focused on new techniques to speed up the assessment of the quality of the developed test programs [26]. Once again, the issue of preliminarily identifying untestable delay faults to reduce the test generation effort and more precisely assess the achieved test effectiveness plays a key role [11]. Clearly, removing functionally untestable faults from the considered fault list allows to increase the achieved coverage, as it is routinely done when adopting standards (e.g., ISO 26262 in automotive) and performing Failure Modes, Effects and Diagnostic Analysis (FMEDA). On the other side, detecting them anyway may increase the overall quality of the product. The experimental results we report in this chapter allow to quantitatively assess the impact of functionally untestable faults and to better understand their origin.

In this chapter we do not focus on the usage of SBST for testing the faults in the CPU, but rather consider the test of a communication peripheral core, building over the techniques overviewed in [26]. We extend them to an interface based on the SPMI^{SM} standard, and analyze the results gathered on a test case where both the scan and SBST solutions were developed. For the first time, comparative results related to a peripheral component are reported with respect to both stuck-at and transition delay faults. An analysis of the results obtained with the two techniques provides the reader with some better understanding of their advantages and limitations. Furthermore, we present an evaluation of SDD coverage of the developed test set.

3 Proposed approach

In order to test a peripheral module within a SoC with a functional approach, such as in SBST, a specific code for the embedded CPU needs to be written, accessing the peripheral registers by means of the system bus. In addition, in case of communication peripherals or modules interacting with the outside of the chip, further stimuli need to come from the external world, i.e., by the ATE. The operations of the embedded microcontroller and the external tester must be synchronized by means of precise timing control or handshake protocols.

For the validation of a SBST set and the assessment of test coverage, a hardware description language (hdl) testbench is used to activate the system and emulate external devices in simulation and fault simulation. The general flow is described in Fig. 2. After the code is written and the testbench is prepared, a functional simulation is performed to ascertain that the Unit Under Test (UUT) performs as planned. Then, fault simulation is required to assess the coverage on a list of faults on the peripheral logic structure.

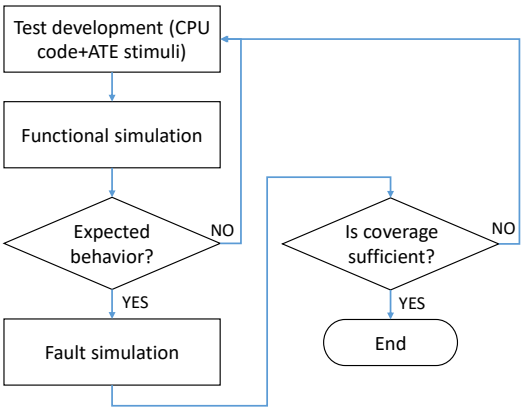

Fig. 2. Test generation flow.

3.1 Generation of the test program and external stimuli

The development of a SBST set usually starts with a series of small program sections able to access each of the peripheral registers and activate all its functionalities (e.g., transmitting and receiving data in different configurations). Any available design validation code can be fruitfully employed in this step. Usually, each part is composed of a preliminary "setup" or configuration phase, and an operational step. It is possible to assume that the available code is already "short enough", i.e., avoids redundant parts, so as not to increase simulation time and also to limit test application duration/costs.

However, two important test-specific points have to be considered. First, the targeted fault model has an impact on the required stimuli: as an example, whereas a

stuck-at test only requires that each register is written and then read first with '0' and then with '1' logic values, when dealing with delay-dependent fault models, such as the transition delay one, the sequence and the timing of operations is fundamental as well. In this case, a sequence of '0'-to-'1' and '1'-to-'0' operations are needed for each node. In this way, stuck-at faults are inherently covered.

Second, while validation usually requires the monitoring of a limited amount of functional results of an operation, in case of test, in order to guarantee high test coverage, the observation of fault effects requires more pervasive data sampling operations. The fault effects need to be propagated either to the outside of the device (to be read by the ATE) or to a bus or memory area readable by the embedded microprocessor.

After a preliminary all-encompassing functional stimulation, if additional coverage is required, it is possible to address the composing elements of the peripheral one at a time, with specific techniques for activating the logic of each part. The most common elements within a digital peripheral are controllers, combinational units such as comparators and algebraic units, sequential devices with a regular structure such as counters, and data buffers.

Controllers are the circuit sections used to handle the control signals regulating the datapath, and are typically implemented using Finite State Machines (FSMs), which are mathematical models of computation. An FSM is composed by a finite number of states; the current state evolves from one to another depending on the external inputs. A test procedure normally aims at activating all possible states and transitions between them, and then making the performed operations visible.

To maximize test coverage in combinational units, available ATPG tools can be fruitfully used to generate a sequence of stimuli on limited parts of the logic. Such sequences need then to be brought to the unit interface by means of microprocessor instructions or external interaction, and then test results have to be propagated to observable points. It may not be always possible to apply any pattern to inner circuitry: this will be further discussed in subsection 3.3.

Regular sequential units such as counters need to be approached taking into account the fact that their test can be quite time-consuming. For this reason, it can be useful to concentrate on applying transitions on the output of each sequential element and propagating them towards observable points, exploiting programmable features. For instance, a 32-bit programmable counter can be set to count in different shorter ranges to activate transitions in all register bits without waiting for 2^{32} clock cycles: this can be accomplished by targeting the elementary increment/decrement operations with the related generation and propagation of carry/borrow bits. Similarly, when testing a data buffer, it is needed to know its characteristics (byte/word accessibility, LIFO/FIFO architecture, etc.) and its implementation in order to develop the most suitable sequence of write and read operations.

3.2 Test Coverage Evaluation

The evaluation of test coverage requires the fault simulation process, i.e., a gate-level simulation reproducing the effect of faults and enabling to determine if the applied stimuli produce a difference between the good and the faulty circuitry. Fault simula-

tion can be performed by suitably instrumenting a model in a logic simulator, and commercial tools are also available. Functional fault simulators aimed at validating fault-tolerant designs and at evaluating the effectiveness of test sets are becoming increasingly popular. A fault simulator may require to provide the sequence of input/output signals at the periphery of the module under test (e.g., in the form of a value change dump – vcd – file), and hence a previous functional simulation run is needed; others may directly handle the complete simulation of the testbench and any other circuit parts not currently addressed for the computation of fault coverage. The latter case, represented by, to name but a few, Cadence Incisive Safety Simulator, Z01X by Synopsys and Silvaco HyperFault, is more convenient for the problem described in this work.

Due to the potentially large number of faults within the logic under test and the non-direct logic monitoring of SBST procedures, which may require many clock cycles to propagate fault effects to observable points, the management and the running time of fault simulation can get critical also when using state-of-the-art tools.

The key factors are the number of observation points and the timing when these are actually sampled. In fact, the simulator will check the observation points only in certain instants (decided by the designer). The more frequent is the check, the larger is the time required by the simulation. On the other hand, a more frequent check may detect a larger number of faults, which may increase the overall speed of the process. This is linked to the algorithm used by the fault simulator: generally, once a fault is excited, the fault simulator creates a new simulation instance to keep track of all the evolutions of the faulty circuit. This simulation instance will be closed once the fault is detected, freeing the resource allocated for that instance (fault dropping). The larger is the number of simulation instances, the higher is the amount of resources required by the fault simulator and consequently the slower is the fault simulation. A similar discussion can be done regarding the number of observation points. The larger is their number, the slower is the simulation, but the higher is the chance to detect a fault. It is important to recall that, in the end, the observation point must be chosen in order to get a coverage indication as close as possible to the real test application; different solutions may be employed within the same flow in order to get a fast albeit approximate information when designing the test and a more precise one at the end. As an example, it may be useful to run experiments while sampling data on all flip-flops, even if they may not be directly observable, to iteratively evaluate the effectiveness of the pattern set in exciting faults (controllability); then, switch to more realistic approaches to improve fault propagation to monitorable points (observability), possibly with minor changes on the test.

Another important point to take into account is related to how to model circuit timing. When simulating the pure logic functionality of a circuit, or when evaluating delay-independent fault models such as the stuck-at or transition delay, zero-delay models are sufficient. In this case, the circuits states are usually updated with an event-driven approach at each clock front, resulting in a relatively low computational effort and therefore in a fast simulation. When verifying timing performance of specific operations, to complement static timing analysis, or when verifying immunity or coverage of delay-dependent faults, such as path delay or small delay defects, precise

timing simulation are required. This approach implies the use of timing data (SDF) and a more complex event scheduler, which, as it will be shown later on, may have a relevant impact on simulation performance.

3.3 Functional Testability

To correctly assess test coverage on a circuit, an important concept has to be introduced. A fault is physically testable if there exists a test for the fault which can be applied on the hypothesis of full accessibility to all circuit nets. Even when using full-scan test approaches, not all input sequences can be applied to the combinational parts of the circuit: therefore, not all faults are testable even under full-scan. For example, a delay fault may not be testable, because no one of the vector pairs able to test it can be applied to the inputs of the combinational block where it is located using LoC and LoS techniques. A fault is functionally testable if there exists a functional test for exciting that fault: when delay testing a circuit using SBST (or during the normal behavior of the system), the signals feeding the addressed path are determined by the program running on the processor and on the stimuli on the interfaces. These impose temporal and spatial correlations among registers/flip-flops and thus among inputs/outputs of the addressed logic. These correlations result in a smaller set of testable faults (Fig. 3).

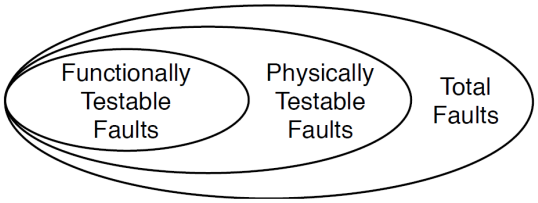

Fig. 3. Fault testability categorization.

Functionally untestable (redundant) faults cannot be activated and/or observed during normal operations of the circuit, therefore they have no impact on circuit behavior and performances. SBST focuses on functionally testable faults, intrinsically avoiding overtesting the circuit's redundant logic [11]. Two definitions are hence used: *fault coverage* is the ratio between tested faults and the total number of faults; *test coverage* is computed using the number of testable faults as denominator.

The identification of functionally untestable faults is still an open problem; however, during the analysis of the peripheral under test and during the generation of the SBST set, the fault list can be pruned to exclude parts of the logic that cannot be functionally operated, e.g., modules deactivated due to hardwired configuration values, any DfT structures that cannot be activated in functional mode, such as scan chain-related signals, or error-handling logic (e.g., redundant paths).

When considering small delay defects, the computation of effectiveness metrics such as DE should take into account the actual possibility of functionally activating the required transition along the minimum slack path, thus increasing the complexity of the problem.

4 Case Study

To demonstrate the feasibility of the functional approach and to analyze its performance with respect to stuck-at and delay faults, a communication peripheral based on the System Power Management Interface (SPMI[SM]) specifications by the MIPI Alliance is used as a case study. The peripheral can handle two-wire serial communications up to 26MHz and includes functions such as bus arbitration, data serialization, error detection and an automated ack/nack protocol.

4.1 Case Study Description

The selected peripheral acts as request-capable slave, i.e., a slave which can initiate sequences on the two-wire SPMI bus (SCLK and SDATA). Fig. 4 shows its basic architecture. The processor system is connected by means of the AMBA AHB bus, and the master/slave AHB interface (equipped with a FIFO mailbox) handles communications. The Control and Status Register (CSR) module includes byte-addressable registers used to control and monitor the peripheral functions. Two finite state machines (FSMs) manage the arbitration (Request FSM) and the general peripheral behavior.

The synthesized peripheral counts about 21,500 equivalent gates and is equipped with full-scan. We underline that scan chains are not used when applying SBST.

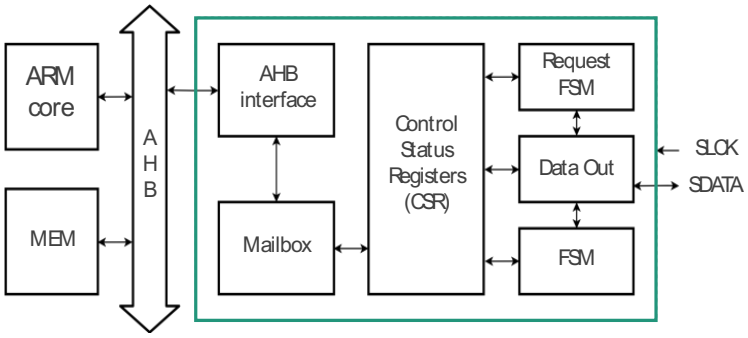

Fig. 4. Architecture of the case study.

4.2 SBST Test Suite

The complete test set is composed of a series of small program/external stimuli sequence pairs, each targeting some specific functionalities or modules within the peripheral:

- *Reset, write 0/1 and read 0/1* (Reset). This segment is operated by the microcontroller, which requests a peripheral reset and then reads the registers. After this, a comprehensive sequence of 0-to-1 and 1-to-0 write operations is done on the CSR, each followed by the needed reads, as in the following pseudo-code

```
// The MCU resets the peripheral (all flip-flops=0)
mcu.SPMI_reset();
for each register in CSR {// write 1 in each register bit
  mcu.ahb.write(reg_addr, 0xFF); // 0->1 transition
  mcu.ahb.read(reg_addr);  // read register content and
                           // store the value in RAM
}
for each register in CSR {// 1->0 transition
  mcu.ahb.write(reg_addr, 0x00);
  mcu.ahb.read(reg_addr);
}
for each register in CSR {
  mcu.ahb.write(reg_addr, 0xFF);
  mcu.ahb.read(reg_addr);
}
mcu.SPMI_reset();
for each register in CSR {
  mcu.ahb.read(reg_addr);
}
```

- *Mailboxes test* (WR-fifo and RD-fifo). For each FIFO buffer (from AHB and from the outside), a sequence of write and read operations is performed to stimulate 0-to-1 and 1-to-0 transitions in the registers, and the "flush" operations are tested as well.
- *Request commands* (Request). In this segment, bus access request commands are programmed to be executed by the peripheral under test, while the external tester will emulate other peripheral on the SPMI bus.
- *All commands* (Commands). In this case, the external tester acts as a bus master and sends a sequence of all possible commands to the peripheral. For read and write commands, all possible payload sizes (1 to 16) are used, and addresses are selected so as to stimulate each bit in the address field with both 0 and 1 values. When writing to the CSR module some care has to be taken to avoid requesting unwanted peripheral operations, by carefully selecting write register addresses and data words (e.g., when writing to *Register 0*). Tests for the authentication mechanism and for activating all states of the state machine are also applied.
- *AHB access control test* (AHB access). This part of the test targets the AHB interface, whose accessible addresses can be programmed. Read/write operations are used aiming at stimulating the address comparators within the module.
- *Counters*. The last part of the test aims at activating the embedded timers for protocol management and timeout condition evaluation.

The results of each segment are read by the CPU, compressed using a software Multiple-Input Signature Register (MISR) sequence and stored in the system memory; then, they are read from the tester with a Read transaction on the SCLK/SDATA pins.

4.3 Fault Simulation

For the case study, Z01X by Synopsys was employed as fault simulator. In general, the tool can be used for two purposes: functional safety assurance, i.e., to check the efficacy of robust design strategies, and for manufacturing assurance, i.e., to evaluate the effectiveness of a functional test set. The synthesized or post-layout circuit netlist in Verilog can be directly simulated using testbenches (also in RTL), libraries and macro models in Verilog and SystemVerilog, thus resorting on the same simulation environment used for design validation.

Three different fault monitoring (strobe) methodologies were compared for coverage and speed:

- *All flip-flops*. Coverage values are computed while monitoring all flip-flops at each clock cycle, as well as the peripheral external I/Os. These data are overestimated since they do not take into account the whole process of fault effect propagation to an observable output, but help evaluating the effectiveness of the test in terms of fault controllability.
- *RAM bus*. Coverage is computed while monitoring transactions on the system RAM, where results are stored after each test operation, and the peripheral external I/Os. The obtained coverage is a good approximation of the one obtainable on the ATE, and the running time is reduced.
- *SDATA*. Coverage is computed monitoring what is sampled by the ATE (i.e., external bus transactions); some coverage is lost with respect to the previous approaches due to the reduced fault observability of the method. This is, however, the slowest and the most memory-intensive methodology.

Regarding the modeling of circuit timing, as most logic simulators, Z01X allows the user to choose how it is handled between the following options:

- *delay mode zero*, ignoring all module path delay information and setting to zero all delay expressions in the code;
- *delay mode unit*, ignoring all module path delay information and converting all non-zero structural and continuous assignment delay expressions to a unit delay of one simulation time unit;
- *delay mode distributed*, ignoring all module path delay information and using distributed delays on nets, primitives and continuous assignments;
- *delay mode path*, deriving timing information from *specify* blocks within the libraries.

Stuck-at and transition delay faults can be analyzed resorting to the simplest timing models, hence increasing fault simulation speed. However, higher precision is needed to handle small delay defects. More realistic data can be derived from static analysis tools, which export cell and path delays taking into account the circuit structures and parasitic elements in SDF files. When importing an SDF file, the simulation complexity increases substantially. Z01X enables setting the size of the delay when injecting transition delay faults with the `+trans+delay+<value>` parameter.

4.4 Experimental results

Table 1 presents the application time required for each of the previously described SBST test segments. The most time-consuming ones are the AHB access test, requiring the application of a large number of patterns for thoroughly testing the combinational logic, and the counter test.

Table 1. Duration of each SBST test segment

Test segment	***Duration [ms]***
Reset	0.926
WR-fifo	0.760
RF-fifo	1.154
Request	1.284
Commands	1.513
AHB access	11.630
Counters	128.390
Total	145.658

Table 2 reports fault simulation results on the stuck-at fault set, which includes 80,640 faults. Among these, at least 11,963 are deemed as functionally untestable, belonging to IP circuitry that cannot be functionally activated in this SoC context, and thus removed from *test coverage* computation. The test segments are applied sequentially on the fault list, with fault dropping. The *CPU time* column reports the duration of fault simulation, performed on an Intel Xeon CPU clocked at 3.00 GHz (a single core is used), while the *Detected* column shows the number of faults covered by the test set. When a test is applied to a sequential circuit, certain faults produce an unknown state at the output when a deterministic result is expected in the fault-free circuit. This condition is known as *potential detection* (numbers in parentheses) and each fault belonging to this category is weighted 0.5 for coverage computation.

It is noteworthy to observe that a significant number of faults produce internal effects on the flip-flops, but cannot be observed on the external circuit outputs, and thus remain undetected. Moreover, the selection of different strobe methodologies may significantly affect the required fault simulation computational effort. The observation on the RAM bus provides a reasonable compromise between accuracy and required CPU time.

Table 3 reports the same data for transition delay faults. In this case, 80,632 faults are considered, out of which 15,452 are functionally untestable. Interestingly, a number of untestable transition delay faults belong to finite state machines, and specifically to transitions from functional to “safe” states corresponding to the *default* branch of case statements of hdl languages, which can be taken only in presence of errors in the circuit behavior.

In order to provide a comparison about the coverage achievable by scan and SBST, another experiment was performed. A scan pattern set was generated with TetraMAX by Synopsys for transition delay and stuck-at faults. The scan test application takes about 120ms with 10MHz shift frequency and at-speed launch/capture, considering a

single scan chain entirely committed to the peripheral under test. Fault coverage is provided for the scan pattern set in the *Scan chains* row of Table 4. In the following rows we report fault coverage for SBST and for the application of both test methodologies in sequence. The total number of stuck-at faults is 80,640, while transition delay faults are 70,207: faults on clock or scan-enable logic are not considered in the latter case, since it is not meaningful to test faults in such logic at functional speeds with scan-based patterns.

Fault simulation shows that the SBST test set uniquely covers 1,335 (1.66%) stuck-at faults and 4,631 (6.60%) transition delay faults in addition to the ones detected by the scan tests. Regarding transition delay faults, the scan test has a higher overall coverage, detecting 9,337 more faults than SBST, but only 6,538 out of these have been classified as functionally testable. Conversely, of the 59,032 detected faults, 5,972 belong to the functionally untestable category.

Results show that, even covering a lower number of faults, the SBST set obtains a better test coverage on transition delay faults in a comparable time. The reader must also note that most of the SBST application time is taken by the Counters segment, which contributes with 2,824 faults to the SBST set coverage, or 361 faults if run after the scan test. In other words, the SBST set is applied after the scan test excluding the Counters segment, it is possible to increase fault coverage by more than 6% (or test coverage by 7%) in 17.27 ms.

SBST test is especially effective, obtaining higher fault coverage than the scan test, on the AHB interface, on the FSMs, on the logic that connects the peripheral to the external world and on the logic used by the Request commands.

By further inspection it is possible to see that most of the logic covered only by the functional test procedure is directly linked to clock gating circuitry or to other functions that are not available while the circuit is in scan-test mode. As a matter of fact, due to the unpredictable functional behavior during shift and capture operations, the circuit is usually brought by hardware to a “safe” state for the application of the scan test, isolating the digital logic from the external world and analog circuitry in mixed-signal devices, and avoiding possible critical configurations of system registers (whose output may be set to fixed values during test). SBST can be fruitfully employed to extend the coverage range of scan test in such cases, even after the circuit is manufactured.

Table 2. SBST stuck-at coverage results with different strobe methodologies

Strobe methodology	***CPU time [s]***	***Detected (potentially)***	***Fault coverage***	***Test coverage***
All flip-flops	72,151	65,720 (1,928)	82.69%	97.10%
RAM bus	32,915	59,796 (1,345)	74.99%	88.05%
SDATA	1,429,180	59,494 (1,346)	74.61%	87.61%

Table 3. SBST delay fault coverage results with different strobe methodologies.

Strobe methodology	***CPU time [s]***	***Detected (potentially)***	***Fault coverage***	***Test coverage***
All flip-flops	88,933	63,142 (73)	78.35%	96.93%
RAM bus	35,968	57,099 (302)	71.00%	87.83%
SDATA	1,761,990	56,748 (310)	70.57%	87.30%

Table 4. Fault coverage of scan, SBST and both tests (strobe on SDATA).

Test	***Stuck-at faults***		***Transition delay faults***		
	Detected (pot.)	***Fault coverage***	***Detected (pot.)***	***Fault coverage***	***Test coverage***
Scan chains	78,562 (0)	97.42%	59,032 (0)	84.08%	81.40%
SBST	59,494 (1,346)	74.61%	56,748 (310)	70.57%	87.30%
Both	79,897 (67)	99.12%	63,663 (42)	90.71%	88.57%

The coverage of small delay defects was evaluated for one of the SBST test segments, *Commands*, which is the one obtaining the highest transition delay test coverage, as it can be seen in Table 5. Obviously, the total coverage is not the simply the sum of the coverage obtained by each segment, since each fault can be covered by more than one test. The following analysis is done using the strobe on the RAM bus and the peripheral external I/Os.

Table 5. Transition fault coverage of each SBST test segment, and according to each strobe methodology.

Test segment	***Transition delay test coverage / strobe methodology***		
	All flip-flops	*RAM bus*	*SDATA*
Reset	44.24%	36.99%	36.15%
WR-fifo	39.07%	30.63%	30.12%
RF-fifo	36.93%	29.71%	29.23%
Request	38.15%	28.24%	27.41%
Commands	51.82%	44.97%	44.09%
AHB access	38.07%	27.83%	27.02%
Counters	38.49%	31.95%	31.43%
Total	96.93%	87.83%	87.30%

As aforementioned, a small delay defect on a logic path can be detected only when its size is larger than the minimum slack. Fig. 5 reports the distribution of minimum slack values on the paths where the faults are located. The histogram shows two distinct peaks, due to paths related to the two different clocks used by the module under test: one corresponds to the AHB clock frequency, synchronizing the control state

machines that interact with the rest of the SoC, and the other to the external SPMI clock which runs at a lower speed.

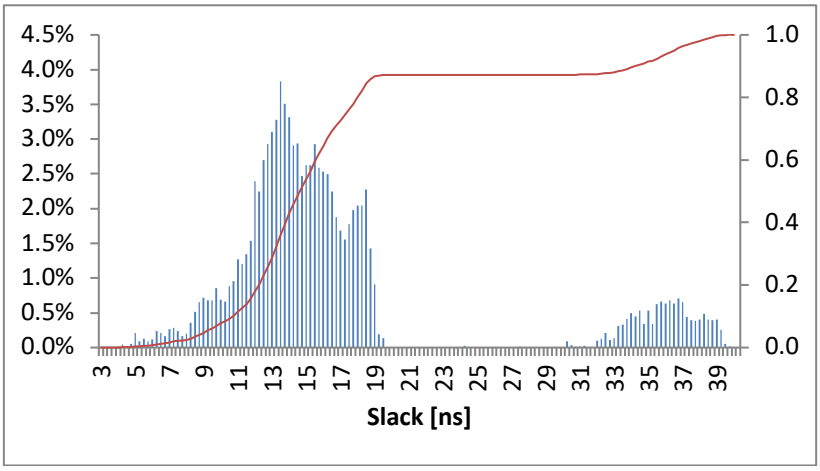

Fig. 5. Minimum slack histogram on the fault paths and cumulative distribution (in red).

Transition delay and SDD coverages, with varying defect sizes, are reported in Table 6. As expected, test coverage rises with larger delay fault sizes; it must be noted that the maximum achievable value with the analyzed test is 44.97%, corresponding to the indefinitely large transition delay fault model. Defect Effectiveness is at all times larger than test coverage; however, its value decreases after the 20 ns delay size. This effect can be due to two causes. Firstly, while test coverage is computed on a fixed number of faults for the all experiments, the DE denominator grows together with fault size, and this can lead to a lower effectiveness even when the absolute number of covered faults is larger. Secondly, since SDDs are emulated in a timing simulation, the presence of hazards and glitches may in some occasions mask the effect of a fault when changing the defect size [28].

CPU time for a fault simulation run is on the order of 12 minutes for transition delay faults, while it rises up to 1.5 hours for small delay faults. This is due to the need of performing more accurate timing simulations using SDF data.

Table 6. Small delay defect coverage of the *Commands* test; the strobe is on the RAM bus.

	SDD fault size					***Transition faults***
	10 ns	*15 ns*	*20 ns*	*25 ns*	*30 ns*	
Detected	1,497	5,315	13,924	14,480	14,554	29,057
Pot. Detected	335	339	370	426	427	515
Not Detected	78,800	74,978	66,338	65,726	65,651	51,060
Fault Coverage	2.06%	6.80%	17.50%	18.22%	18.31%	36.68%
Test Coverage	2.55%	8.41%	21.65%	22.54%	22.66%	44.97%
Defect Effectiveness	3.81%	17.79%	27.93%	26.34%	25.97%	--

5 Conclusions

This chapter describes a case study corresponding to a peripheral module within a SoC, for which a test for both stuck-at and transition delay faults has been developed resorting to the scan approach and to a functional one, based on SBST. We outlined a specific approach to develop the latter test targeting both stuck-at and transition delay faults. Extensive results have been presented, showing that the two methods have different and complementary characteristics. While scan test generation is fully automated, the functional test must be manually built. The fault coverage achieved by scan is higher, but some faults (especially numerous when considering delay faults) are only detected resorting to the functional approach. Moreover, we showed that some of the faults which are only detected by scan are functionally untestable. Hence, scan is likely to produce a higher degree of overtesting. We also reported some preliminary experimental results about the coverage that the functional approach can provide with respect to Small Delay Defects. Our results may allow test engineers to better understand the impact of functionally untestable faults on the achieved yield, reliability and quality of the product. We discussed the above points, providing examples for each category.

Work is currently being done in order to further improve the method to develop functional test programs targeting delay defects. Moreover, we are working to devise solutions to identify functionally untestable faults, extending to peripheral modules some of the ideas proposed in [29].

Acknowledgements. The authors wish to thank Andrea Casalino and Calogero Brucculeri for helping in the setup of the experimental campaigns.

References

1. M. Psarakis, D. Gizopoulos, E. Sanchez, M. Sonza Reorda, “Microprocessor Software-Based Self-Testing”, IEEE Design & Test of Computers, 2010, Vol. 27, Issue: 3, pp. 4-19
2. P. Bernardi, R. Cantoro, S. De Luca, E. Sanchez, A. Sansonetti, “Development Flow for On-Line Core Self-Test of Automotive Microcontrollers”, IEEE Transactions on Computers, 2016, Volume: 65, Issue: 3, pp. 744-754
3. A.-U.-R. Shaheen, F.A. Hussin, N.H. Hamid, N.H.Z. Ali, “Automatic generation of test instructions for path delay faults based-on stuck-at fault in processor cores using assignment decision diagram”, IEEE International Conference on Intelligent and Advanced Systems (ICIAS), 2014, pp. 1-5
4. V. Singh, M. Inoue, K.K. Saluja, H. Fujiwara, “Instruction-based delay fault self-testing of processor cores”, IEEE International Conference on VLSI Design, 2004, pp. 933-938
5. N. Hage, R. Gulve, M. Fujita, V. Singh, “On Testing of Superscalar Processors in Functional Mode for Delay Faults”, IEEE International Conference on VLSI Design and International Conference on Embedded Systems (VLSID), 2017, pp. 397-402

6. M. Psarakis, D. Gizopoulos, M. Hatzimihail, A. Paschalis, A. Raghunathan, S. Ravi, “Systematic Software-Based Self-Test for Pipelined Processors”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2008, Vol. 16, N. 11, pp. 1441-1453
7. K. Christou, M.K. Michael, P. Bernardi, M. Grosso, E. Sanchez, M. Sonza Reorda, “A Novel SBST Generation Technique for Path-Delay Faults in Microprocessors Exploiting Gate- and RT-Level Descriptions”, IEEE VLSI Test Symposium, 2008, pp. 389-394
8. C.H.-P. Wen, L.-C. Wang, K.-T. Cheng, K. Yang, W.-T. Liu, J.-J. Chen, “On a software-based self-test methodology and its application”, IEEE VLSI Test Symposium, 2005, pp. 107-113
9. W.-C. Lai, A. Krstic, K.-T. Cheng, “Functionally testable path delay faults on a microprocessor”, IEEE Design & Test of Computers, 2000, Vol. 17, N. 4, pp. 6-14
10. M. Fukunaga, S. Kajihara, S. Takeoka, “On estimation of fault efficiency for path delay faults”, IEEE Asian Test Symposium, 2003, pp. 64-67
11. P. Bernardi, M. Grosso, E. Sanchez, M. Sonza Reorda, “A Deterministic Methodology for Identifying Functionally Untestable Path-Delay Faults in Microprocessor Cores”, IEEE International Workshop on Microprocessor Test and Verification, 2008, pp. 103-108
12. S.K. Goel, K. Chakrabarty, “Testing for Small-Delay Defects in Nanoscale CMOS Integrated Circuits”, CRC Press, 2017
13. M. Grosso, S. Rinaudo, A. Casalino, M. Sonza Reorda, “Softare-Based Self-Test for Transition Faults: a Case Study”, IEEE International Conference on Very Large Scale Integration (VLSI-SoC), 2019, pp. 76-81
14. P. Nigh, A. Gattiker, “Test Method Evaluation Experiments & Data”, IEEE International Test Conference, 2000, pp. 454–463
15. E.S. Park, M.R. Mercer, T.W. Williams, “Statistical Delay Fault Coverage and Defect Level for Delay Faults”, IEEE International Test Conference, 1988, pp. 492–499
16. R. Mattiuzzo, D. Appello, C. Allsup, “Small-delay-defect testing”, Test & Measurement World, June 2009, pp. 37-41
17. C. Metzler, A. Todri-Sanial, A. Bosio, L. Dilillo, P. Girard, A. Virazel, “Timing-aware ATPG for Critical Paths with Multiple TSVs”, IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems, 2004, pp. 116-121
18. M. Yilmaz, M. Tehranipoor, K. Chakrabarty, “A Metric to Target Small-Delay Defects in Industrial Circuits”, IEEE Design & Test of Computers, Mar.-Apr. 2011, Vol. 28, N. 2, pp. 52-61
19. A. Uzzaman, M. Tegethoff, Bibo Li, K. Mc Cauley, S. Hamada, Y. Sato, “Not all Delay Tests Are the Same - SDQL Model Shows TrueTime”, in IEEE Asian Test Symposium (ATS), 2006, pp. 147-152
20. M. Bushnell, V. Agrawal, “Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits”, Kluwer Academic Publisher, 2000
21. K.-T. Cheng, H.-C. Chen, “Classification and identification of nonrobust untestable path delay faults”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1996, Vol. 15, N. 8, pp. 845-853
22. X. Liu, M.S. Hsiao, “On identifying functionally untestable transition faults”, IEEE International High-Level Design Validation and Test Workshop, 2004, pp. 121-126
23. Y. Zhang, Z. Peng, J. Jiang, H. Li, M. Fujita, “Temperature-aware software-based self-testing for delay faults”, IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015, pp. 423-428
24. A. Touati, A. Bosio, P. Girard, A. Virazel, P Bernardi, M. Sonza Reorda, “Improving the Functional Test Delay Fault Coverage: A Microprocessor Case Study”, IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2016, pp. 731-736

25. A. Touati, A. Bosio, P. Girard, A. Virazel, P. Bernardi, M. Sonza Reorda, “Microprocessor Testing: Functional Meets Structural Test”, World Scientific Journal of Circuits, Systems, and Computers, 2017, Vol. 26, N. 8, pp. 1-18
26. A. Floridia, E. Sanchez, M. Sonza Reorda, “Fault Grading Techniques of Software Test Libraries for Safety-Critical Applications”, IEEE Access, 2019, Vol 7, pp. 63578-63587
27. A. Apostolakis, G. Gizopoulos, M. Psarakis, D. Ravotto, M. Sonza Reorda, “Test Program Generation for Communication Peripherals in Processor-Based SoC Devices”, IEEE Design & Test of Computers, 2009, Vol. 26, N. 2, pp. 52-63
28. J. Wang, H. Li, Y. Min, X. Li, H. Liang, “Impact of Hazards on Pattern Selection for Small Delay Defects”, IEEE Pacific Rim International Symposium on Dependable Computing, 2009, pp. 49-54
29. R. Cantoro, S. Carbonara, A. Floridia, E. Sanchez, M. Sonza Reorda, J.-G. Mess, “An analysis of test solutions for COTS-based systems in space applications”, IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), 2018, pp. 59-64