J Electron Test (2012) 28:189–200
DOI 10.1007/s10836-012-5287-2

Software-Based Testing for System Peripherals

M. Grosso · W. J. Perez Holguin · E. Sanchez · M. Sonza Reorda · A. Tonda · J. Velasco Medina

Received: 24 October 2010 / Accepted: 5 February 2012 / Published online: 26 February 2012

Abstract Software-based self-testing strategies have been mainly proposed to tackle microprocessor testing, but may also be applied to peripheral testing. However, testing system peripherals (e.g., DMA controllers, interrupt controllers, and internal counters) is a challenging task, since their observability and controllability are even more reduced when compared to microprocessors and to peripherals devoted to I/O communication (e.g., serial or parallel ports). In this paper an approach to develop functional tests for system peripherals is proposed. The presented methodology requires two correlated phases: module configuration and module operation. The first one prepares the peripheral to work in the different operation modes, whereas the second one is in charge of exciting the whole device and observing its behavior. We propose a methodology that guides the test engineer in building a compact set of test programs able to reach high structural fault coverage levels in a short time. Experimental results demonstrating the method effectiveness for two real-world case studies are finally reported.

Keywords System peripheral · Functional testing · SBST · DMA controller · Interrupt controller

Responsible Editor: C. Metra

M. Grosso · E. Sanchez (✉) · M. Sonza Reorda · A. Tonda
Dipartimento di Informatica e Automatica, CAD Group,
Politecnico di Torino,
Torino, Italy
e-mail: ernesto.sanchez@polito.it

M. Grosso
e-mail: michelangelo.grosso@polito.it

M. Sonza Reorda
e-mail: matteo.sonzareorda@polito.it

A. Tonda
e-mail: alberto.tonda@polito.it

W. J. Perez Holguin · J. Velasco Medina
Bionanoelectronics Group, Universidad del Valle,
Cali, Colombia
e-mail: wjperezh@univalle.edu.co
e-mail: wilson.perez@uptc.edu.co

J. Velasco Medina
e-mail: jvelasco@univalle.edu.co

W. J. Perez Holguin
GIRA Group, Universidad Pedagógica
y Tecnológica de Colombia,
Sogamoso, Colombia

1 Introduction

S*ystem peripherals* are devices of paramount importance in processor-based systems because they provide essential support to the processor elaboration tasks with system services, which are clearly distinct from the pure communication services supplied by I/O peripherals. Typical examples of system peripherals include DMA controllers, interrupt controllers, and timers.

Often these modules are deeply embedded in Systems on Chips (SoCs), or exist as stand-alone devices included in processor-based systems. In the first case, system peripheral accessibility is particularly critical, while in the second case mature methodologies based on the inclusion of expensive devices (e.g., In-Circuit Testers) mainly oriented to test application after board manufacturing and assembly may critically limit device performance. In both cases, the test activities represent a challenging task, either to support end-of-line manufacturing test, or incoming inspection [4]. Currently, SoCs can include a large amount of hardware resources embedded in a single chip, thus allowing the development of powerful and inexpensive systems in relatively short times.

The most common devices composing a SoC are processor cores, memory modules, bus interfaces, peripherals, and customized logic modules. The trend has stimulated the development of a large number of applications that exploit this design paradigm, leading in turn to the rise of new challenges for test designers: it is now necessary to tackle large circuits, modules with very reduced accessibility, cores from third parties and parts with different design styles.

Different hardware-based and software-based testing methods have been proposed to tackle the testing issues mentioned above. In the former case, the proposed methods mainly introduce some changes in the circuit to support testing activities (Design for Testability, or DfT): scan testing and BIST are among the most commonly employed methods in this group [5]. When dealing with SoCs, hardware-based testing approaches can be applied in a relatively easy way, and scale well with circuit complexity; however, usually these techniques also introduce a significant area and performance overhead, and sometimes require unacceptably long test times. In addition, scan-based methodologies can hardly support at-speed testing, and test application with a reduced clock speed may considerably decrease the defect coverage of the applied test vectors [19] as well as impair coverage figures with respect to delay-related faults. Moreover, hardware-based testing requires the capability to modify core models, which is often a prohibited task in several design flows, or when dealing with incoming inspection.

It is well known that digital devices are usually thoroughly tested by the manufacturer at the end of the line of production. However, Original Equipment Manufacturer (OEM) companies often decide to perform a further test (called *incoming inspection*), to check for defective components before using them in their manufacturing process, with the purpose of ensuring a higher quality of the final product. However, OEM companies hardly have access to any information about the device internal structure and thus are not able to generate a test using structural methodologies. This lack of information makes testing techniques based on structural information unapplicable when tackling incoming inspection, and forces the adoption of purely functional techniques.

In several cases (e.g., when used in safety-critical applications), electronic devices are required to be checked for possible faults even during their operational life (*on-line test*). This kind of test may exploit structural and DfT-based test solutions, when detailed information about the device are available, or resort to a purely functional approach when this is not true.

In all the above scenarios, one of the least intrusive and most promising methods for circuit testing exploits the processor embedded in the SoC or available in the system, to execute suitable sets of test programs. Such test programs are specifically designed to test the processor and other modules accessible by it, without any change in the hardware. This method, often called *Software-Based Self-Test* (SBST) [21], has several advantages with respect to hardware-based methods, such as:

- the testing procedure can be conducted with very limited area overhead, if any;
- the average power dissipation is comparable to the one observable in mission mode;
- the possibility of damages due to excessive switching activity existing for other methods is eliminated;
- it can be run at the maximum system speed, thus allowing testing of a larger set of defects, including delay-related ones;
- it is applicable even when the structure of a module is not known or cannot be modified;
- it intrinsically targets the faults that can be excited in the system during its functional mode of operation, only, therefore minimizing overtesting issues.

On the downside, SBST methods require the generation of effective test programs, i.e., assembly programs (and possible associated stimuli) able to reach high fault coverage levels comparable with those obtained by applying hardware-based test methodologies. Such test programs are often difficult to obtain, and their development may require a good knowledge of the functional behavior of the Device Under Test (DUT).

SBST methodologies have been largely explored to test microprocessor cores [21]. The SBST methods basically derive from ideas introduced in literature some decades ago [23], but exploit the most recent advances in test program generation techniques. Roughly speaking, the most representative SBST methodologies can be classified according to the processor description used during the generation flow. The possibilities vary from processor high-level or functional descriptions to lower level representations, such as RTL and gate-level netlists. Apart from the capacity of each method regarding the obtained test coverage, its global effectiveness must be evaluated taking into account its scalability and ease of automation. In the following, some of the most important research works are described.

Relying only on the processor Instruction Set Architecture (ISA), the authors in [22] propose VERTIS, a tool that produces for every processor instruction a lengthy sequence of code counting with random selected operands in order to excite as completely as possible the instruction particularities, thus obtaining a very large test set. Its effectiveness was assessed on the GL85 microprocessor, resulting in 90.20% stuck-at fault coverage, which was much higher than the one obtained through traditional sequential Automatic Test Pattern Generator (ATPG) tools.

A different approach is described in [20]: the FRITS (Functional Random Instruction Testing at Speed) approach generates random instruction sequences with pseudorandom data generated through a software-implemented LFSR. 70%

stuck-at fault coverage is reported on the Intel Pentium® 4 processor, applying a fully self-contained approach where the test program is stored in the cache and no bus operations are executed, without additional controllability or observability needed on the buses.

The approaches described in [[18], [6]] are based on *constrained test generation*. In these cases, the processor addressed module is described at structural level, while the rest of the processor is described at a higher abstraction level—providing the constraints for pattern generation. The effort of the ATPG is hence reduced since the automatic tool faces a circuit with low complexity. Specifically, in [6] for every single module, a preliminary set of instruction templates is generated. The actual goal of the generated templates is to easily provide the module with the appropriate input vectors, as well as to be able to observe module responses. Thus, considering module observability and controllability the most suitable templates are selected, and then, a constrained ATPG exploiting the selected templates is in charge of test vector generation, and the resulting vectors are finally translated to sequences of instructions. The described method achieves 95.2% stuck-at fault coverage on the Xtensa processor core (a 5-stage pipelined configurable RISC processor).

The authors of [16] developed a technique applicable to stuck-at and path-delay fault testing. They build a set of spatial and temporal constraints in the form of Boolean equations from the microprocessor ISA which are able to guide the generation of instruction sequence during the application of an ATPG or a random-based generator to specific processor modules. Following the proposed technique, 92.42% stuck-at fault coverage is reported on the Parwan processor core.

Other methods exploit a combination of processor abstraction models, such as RTL descriptions and ISA specifications. The deterministic algorithms described in [15] take advantage from information about the functions performed by the different addressed modules, and produce test programs by following some guidelines they introduce to target specific components. The resulting test programs are loop-based pieces of code that deterministically provide the modules under testing with a series of data inputs carefully selected. The method proposed in [15] is evaluated on three different synthesized versions of the PLASMA processor core achieving about 95% stuck-at fault coverage, demonstrating the method feasibility. Later, in [9], the authors extend their previously introduced technique for more complex pipelined processor cores (miniMIPS and OpenRISC 1200), taking into account the analysis of data dependencies of available SBST programs and general parameters of the pipeline architecture and memory system.

A different approach is presented in [7], the described method exploits a simulation-based framework that utilizes an evolutionary algorithm as automatic test program generator. Roughly speaking, an evolutionary algorithm is a population-based optimizer that imitates the natural process of biological evolution in order to iteratively refine the population of individuals (or test programs) by mimicking the Darwinian evolution theory. The approach effectiveness was demonstrated by comparing its performance with a pure instruction randomizer, both working on a RTL description of the LEON2 processor.

It should be noted that the research community has devoted most of the efforts on the test of processor cores, while peripheral cores testing has been somehow neglected. In modern systems, there is a steady increase in the number and complexity of integrated peripherals, allowing them to include new and enhanced features; however, their increased complexity makes testing of system peripherals a challenging task.

In the following, brief descriptions of some of the most important approaches that tackle peripheral testing resorting to SBST techniques are first presented. Dealing with I/O peripherals testing, the authors in [22] proposed a method to be implemented basically by hand, that mainly relies on the experience of the test engineer, who sequentially maximizes the various coverage metrics, generating one or more test programs for every metric. This process is repeated until sufficiently high coverage values are obtained for all the chosen metrics. The approach is validated on an 8251-compatible USART, showing rather low stuck-at fault coverage (67.89%).

In [3] a fully automated methodology for the generation of test programs for I/O peripheral cores (PIA, UART) embedded in a SoC, is presented. The methodology is based on the exploitation of the correlation between high-level metrics (e.g., Toggle, Expression, Condition, Branch and Statement Coverage) and the gate-level fault coverage. The generation loop is driven by an evolutionary optimizer able to produce and improve *test blocks* that properly configure and activate the considered device.

In [1] the authors describe a generic and systematic flow of SBST application on two communication peripheral cores. The methodology achieves high fault coverage but needs a deep knowledge of the peripheral core, leading to long test development time with a high human effort.

In [2] the authors mixed the methodologies presented in [3] and [1] to propose a solution mainly based on two phases: in the first one, the peripheral is suitably configured; then, in the second one, it is correspondingly activated, and its behavior is observed. If the peripheral is intended to support the communication between the SoC and the outside world, the latter phase requires the coordinated excitation and/or observation of the peripheral external ports, while the SoC reads/writes the data received/transmitted by the peripheral. This goal can be achieved by resorting either to an external Automatic Test Equipment (ATE), or to a loop-back mechanism. In the proposed approach, the test sets for the individual subcomponents of the communication peripheral core are pre-computed and pre-generated. The proposed methodology has been evaluated on a sample SoC with three popular communication peripherals (a UART, an HDLC and an Ethernet controller).

In [14] Hwang and Abraham developed a bus-based test architecture and applied it to an ARM-based SoC with two ISCAS89 cores, a USB Controller and a Viterbi encoder. In [13] the authors proposed a SBST approach for various ISCAS89 cores providing results for stuck-at and path delay fault models, while the approach in [17] uses the mechanism proposed in [13] in a DLX-based SoC to apply scan vectors in four ISCAS89 cores.

Concerning system peripheral testing, in [8] the authors propose a semi-formal methodology to generate test sets targeting specific behaviors, often called *corner cases*: a composition of border behaviors for different design parts or blocks of the device under test. Initially, the device is modeled in a simplified manner in a process called abstraction. A proper set of signals is carefully selected, and the model is then translated into a FSM where each possible combination of their values corresponds to a state. A coverage-driven test generator produces a set of abstract tests containing for every case a sequence of states. The number of reached states is used as a metric. Finally, every single abstract sequence of states is translated in order to obtain the real test set. The presented methodology is exploited for the validation of a Direct Memory Access controller (DMAC) embedded in the ST50, a RISC-based microcontroller. The authors report a statement coverage figure of about 87%, and a branch coverage of 75% in the VHDL description of the device at the RT level.

The approach initially presented in [11] aims at system peripherals verification, and it has then been extended to testing [12]. These works propose a method to develop functional tests for DMA controllers embedded in SoCs. The proposed approach deterministically creates a compact set of configurations for the DUT able to obtain high coverage figures firstly for verification purposes, and then for testing. Experimental results for an 8051-based benchmark SoC with an embedded 8237-compliant DMA controller core were reported to demonstrate the method effectiveness.

Differently from the previous approaches, in this paper a complete framework for the end-of-production testing of system peripheral devices is proposed. The methodology is suitable for embedded peripherals as well as for stand-alone ones, and improves the deterministic methods presented in [11, 12] by providing additional guidelines for achieving high-coverage functional test stimuli starting from high-level descriptions of the addressed module. Beyond the analysis and stimulation of the operating modes of the selected peripheral, the paper gives details about how to thoroughly excite and observe faults in the most widespread sub-modules that can be found in such devices, by generating a compact set of functional stimuli. Additionally, the graph visiting algorithm has been improved.

The rest of the paper is organized as follows: the proposed approach is described in Section 2; Section 3 presents relevant information about two widely adopted commercial system peripherals: an 8237-compliant DMA controller, and the LEON3 interrupt controller (IRQ). Section 3 is concluded showing the obtained results that experimentally validate the proposed methodology. Finally, Section 4 concludes the paper.

2 Proposed Approach

This paper presents a new methodology for system peripheral testing. Its main contributions are:

- a general framework for creating SBST programs, based on module configuration and operation, to thoroughly sensitize and observe faults in the targeted peripheral;
- a methodology for the deterministic generation of an effective and compact set of configurations for the device under test, which are the basis for the successive development of actual test programs;
- a collection of practical guidelines for the development of test programs aimed at exciting the control logic and the most common elaboration modules composing system peripherals and observing the produced results, starting from the previously developed set of configurations.

By following the described workflow, a compact set of test programs targeting static faults is generated, based on a functional, high-level description of the device under test. The needed system description mainly corresponds to the specifications of the selected module, from where the operational modes, the configuration registers and an architectural block diagram are usually available. The effectiveness of the generated test programs has to be evaluated exploiting fault simulation, but higher-level metrics (e.g., based on HDL code coverage), when available, may as well provide a first and faster feedback on the activation of the different components of the addressed system peripheral in early development stages. In case the obtained set of programs does not reach the desired fault coverage, the configuration/test program generation process may be iterated while selecting alternative configuration combinations and increasing the amount of elaborated data.

2.1 Test Application Requirements

Figure 1 shows the conceptual view of the setup for applying the SBST methodology to a system peripheral. An available processor core in the system is exploited to run a suitably generated test code that controls the addressed system peripheral behavior. This code has to be stored in a memory module

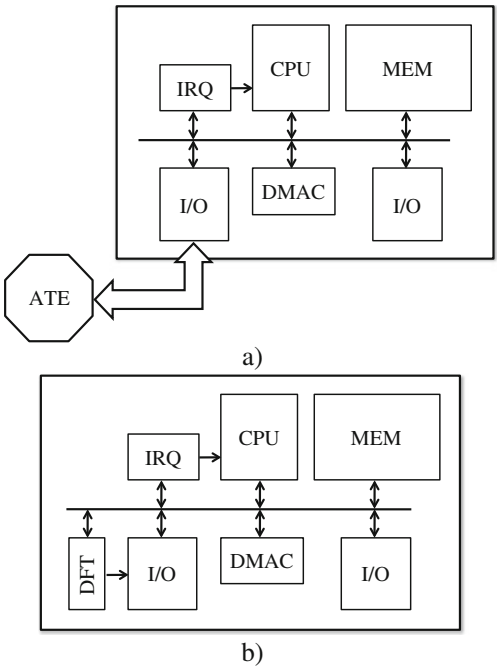

Fig. 1 Testing architectures **a** Including ATE **b** Including DfT facilities

and includes both system configuration and operation routines able to activate the targeted peripheral module and then check the correct test outcome. Furthermore, I/O peripheral devices take part in the test process to stimulate the input ports of the addressed module which are not directly controllable by the processor.

For the purpose of this paper we only focus on possible faults in system peripherals, and assume that all involved memory, processor and I/O peripheral cores are fault-free. The proposed methodology is aimed at providing tests able at activating peripheral faults and making their effects observable by means of the system microprocessor.

An external Automatic Test Equipment can provide additional stimuli to I/O peripherals involved in the test operations (Fig. 1a). As an example, a simple DMA controller test may be based on the data transfer between an available I/O peripheral and data system memory: the CPU configures the transfer, and then appropriate inputs have to be applied by the ATE to the I/O peripheral to start the communication protocol and to activate the DMA module. Finally, the CPU checks the content of the data memory to observe the test results.

Alternatively, integrated debug structures or Design for Testability (DfT) modules in the system architecture may be used to obtain direct controllability and observability on the system peripherals from the processor core in a suitably defined test mode, without resorting to external ATEs (Fig. 1b). In the aforementioned example, I/O peripheral operations may be controlled by a processor-accessible register interfaced to the I/O module itself.

For the purpose of this paper, we assume that either an external ATE, or an integrated DFT module is available, and focus on the generation of the suitable test stimuli, only.

2.2 System Peripheral Test Execution

The proposed SBST methodology for system peripherals is based on the stimulation of the module under test exploiting an available processor core that runs a suitably developed set of test programs and applies carefully devised data patterns. Each test program stimulates the targeted module in a specific configuration. After the test program has been uploaded into the system code memory, the test application phase is based on the following four main steps:

- Initial configuration of supporting elements
  The processor core in this step is devoted to prepare the components, different than the targeted system peripheral, that support the application of the considered test. For example, when considering the case of a DMA controller (DMAC), it is indispensable to initialize involved memories and I/O peripherals with adequate values, depending on the requirements of the testing case. When dealing with an interrupt controller, the components that must actually generate interruptions are initialized to accomplish special actions derived from the testing requirements.
- System peripheral configuration
  The processor core executes appropriate instructions to configure the targeted system in a specific operating mode. The proposed methodology for the generation of an effective and compact configuration set is described in subsection 2.3.
- Test execution (system peripheral operation)
  The test is started and suitable stimuli are provided to the system peripheral under test according to the defined configuration settings.
- Monitoring of observable points
  In this final step, the processor core compares the obtained results with the expected ones, in order to check test outcomes.

It is important to mention that the two final steps may be performed concurrently, depending on the system peripheral architecture and requirements. For example, in the case of the DMAC, the microprocessor core controls observable points guaranteeing not only that correct data have been transferred but also that they come in the expected order; this process is performed at the end of the test application. On the other hand, when considering interrupt controllers, testing priority mechanisms may require the use of interrupt service routines, whose execution is triggered by the peripheral itself: in this way, excitation and monitoring are performed in parallel, making the test shorter and more effective.

2.3 Generation of the Configuration Set

In this subsection we provide a brief description of the method proposed for the generation of an effective and compact configuration set. This method derives from a procedure initially developed for verification purposes that has been extended to the testing field by incorporating additional considerations concerning testing issues, such as those related to controllability and observability. The methodology consists of the following main steps:

- Analysis of the available high level description of the peripheral under test, to identify the device configuration registers and to select the most suitable metrics to guide the test generation process. Whenever available, high level metrics, such as the ones based on code coverage, can be exploited to assess the exhaustiveness of the configuration set.
- Identification of *shared* and *unshared* resources, depending on their role in the system peripheral. Shared modules are either devoted to coordinate the functions of the system, or unique along the data path. Unshared modules are represented by a set of instances described by the same block of HDL code; usually, these modules carry out the same elaboration function.
- Building of the *configuration graph*, whose main goal is to provide a well organized structure that directs the test generation process, allowing a reduction in the number of configurations to be applied for verification and testing purposes, and thus avoiding unnecessary code replication. In order to describe graph construction, it is advantageous to define *configuration bits* as bits that control a certain functional characteristic of a specific device resource. Such bits can be found throughout all registers of the DUT and are grouped according to the functionality they control. Conceptually, the configuration graph is a directed graph, whose nodes are grouped in levels, where all nodes in the same level correspond to a group of configuration bits. Different nodes in the same level specify (mutually exclusive) modalities of a specific device feature. An arc in the graph from node A to node B, which belong to two different levels, exists iff the device can functionally be configured assigning to the groups of configuration bits the values represented by nodes A and B. A path obtained traversing the graph from top to bottom univocally defines a specific configuration mode for the addressed system peripheral. Figure 2 shows an example of a configuration graph.
- Extraction of the *configuration paths*, whose main goal is to generate the set of device configurations from which actual test programs will derive. A *visiting algorithm* is exploited to extract the configuration paths from the previously generated configuration graph. Theoretically, to test all configurations of the DUT, each arc of the graph should be traversed by at least one path; experimental evidence, however, suggests that it is sufficient to cover all nodes to obtain satisfactory results. In [11], where the goal was verification, the coverage requirement for nodes belonging to unshared modules could be relaxed, to reduce the total number of paths: more in detail, the requirement becomes that each resource configuration mode needs to be activated in at least one of the corresponding unshared modules. As an example, to perform functional verification in a DMA controller with 4 channels described by the same HDL code, the transfer count register can be activated only on 1 of the 4 channels. Differently from that, for testing purposes each configuration mode needs to be tested in every unshared module in order to obtain complete test coverage, since even if the modules derive from the same HDL description, they are physically different. For this reason, a higher number of paths is normally required to provide a useful test set. Thus, the goal of the visiting algorithm is to include each node of the graph in at least one path while keeping the total number of paths as low as possible. Each created path identifies a configuration of the system peripheral. In addition, there can be both *compulsory* paths specified by the verification engineers and *prohibited* configurations inherent to the DUT.

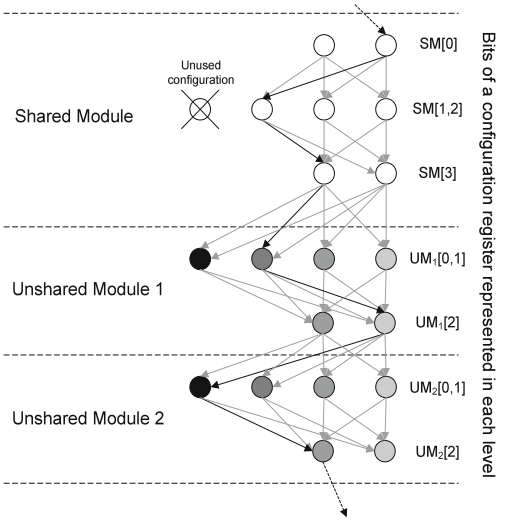

Fig. 2 Configuration graph example. Bits on the right are classified according to the module they are able to configure within the peripheral. The grey arcs represent all possible configurations of the device. The arcs in black show a configuration path

From each of the above generated configurations (i.e., paths), a test fragment can be derived, where the processor is used to initialize all the components involved in the test, execute the operational part, and elaborate the obtained results

in order to guarantee the observability of the whole process. Further details about this step are provided in the next subsection.

2.4 Test Program Development Guidelines

An effective test program suite for system peripherals has to excite faults and propagate their effects to observable points. In order to efficiently stimulate the targeted module in functional operation mode, it has to be activated in different configurations, and in each configuration the inner resources need to be effectively stimulated. Test procedure generation is a non-trivial task that generally relies on architectural device and system knowledge; in this subsection, we discuss the development of effective test programs starting from the generated configuration set.

From every configuration developed through the methodology described in subsection 2.3, a test program is derived targeting specific functional modes. A quick analysis of the configuration provides hints for the development of a functional code fragment targeting a subset of the peripheral inner resources, and which may correspond, e.g., for a DMA controller, to one or more parallel data transfers or, for an Interrupt controller, to an interrupt arbitration sequence. Each specific case needs a defined sequence of operations to be programmed and performed by the system processor and by other system components and/or by an external ATE, and an effective set of data to be processed. This may imply filling the system memories with suitable data values, programming I/O peripherals, or emulating system communications.

In addition, error condition monitoring has to be implemented in the test fragment, which usually corresponds to additional memory or I/O operations.

Determining the most effective means of testing the different modules composing a system peripheral is an integral part of the proposed approach. The following paragraphs present some guidelines for the most common modules that can be found in system peripherals.

- Registers

  The best way to detect faults in registers is clearly writing the register bit with different values, and then reading the register content. Thus, a suitable SBST program oriented to system registers test may be composed of a couple of operations able to toggle register bits, followed by the appropriate reading instructions that guarantee the required observability. Since it is likely that configuration registers are involved in almost all the configuration paths, reading and writing operations can be easily included in different test fragments.
- Counters

  When tackling counters of n bits, a task requiring the complete counting process in the available range (i.e., 2^n operations) is a very good option to detect all possible faults in the DUT. However, shorter test sequences can be experimentally determined to reach the same target. As an example, a more effective strategy that employs a sequence of only $2n$-1 steps of *shift*, *decrement* and *read* operations on the counter register can be used. Let us consider a 4-bit down-counter: first, the value '0001b' is assigned to the initialization register; then, the value is *decremented* once and subsequently *read*. The starting value is then *shifted* to the left, loading in this case '0010b', and the operation repeated. The reading performed after the decrement allows the detection of possible faults. Afterwards, the same procedure needs to be repeated starting from the '0011b' configuration. It can be experimentally proved that the same fault coverage on sequential and combinational elements is obtained as with a full-range count, trading count cycles for register write/read operations. Considerations like this become even more important for reducing test time when the addressed modules may perform different operations, e.g., in the case of a up/down counter which would require two times the number of steps to be completely checked.
- Priority encoders

  Priority encoders are often used to resolve conflicts due to contemporary requests for the same resource. A test sequence for this kind of components can be based on generating a series of parallel requests, whose specific arrangement can be easily derived exploiting combinational ATPG algorithms. While devising a test for a priority controller, it is important to involve all the related unshared modules, in order to uncover faults at the interface between them and the priority logic. Targeting priority logic requires devising a well coordinated testing scheme that properly configures supporting elements in order to produce the correct sequence of stimuli able to activate the priority mechanisms in the expected conditions. Then, an observability phase is activated by the processor through code routines (e.g., Interrupt Service Routines, or ISR), which must be able to identify if the priority logic answered in the appropriate order to the provided stimuli. As an example, to check the correct prioritization performed by an interrupt controller, the following approach may be used: first, a memory checkpoint variable is reset by the processor. Then, while the processor executes an idle loop, simultaneous interrupt requests are activated. The ISR associated to the highest priority request, and only that one, is able to set the checkpoint variable and guarantee the correct behavior of the prioritization module. Therefore, checking the value of the variable allows easily controlling whether the correct ISR has been activated by the Interrupt Controller.

Additional details are given for specific conditions in Section 3.

3 Case Studies

The proposed methodology was experimented on embedded peripherals belonging to two different integrated systems. The first one is built around an Intel 8051-compatible microprocessor core: in this case the target is an 8237-compliant DMA controller. The second one is based on the LEON3 architecture and in this case the target device is the Interrupt controller.

To obtain effective test program sets, the proposed generation methodology is applied in two consecutive phases. The first aims at saturating high level metrics, and delivers a basic test bench whose main goal is to thoroughly excite device functionalities exploiting the configuration set obtained from the configuration graph. This test set can be successfully employed as a starting point for test. In the following phase, test-related considerations are applied to provide additional test programs and improve the obtained fault coverage.

All data about coverage metrics have been gathered at the end of the simulation process performed on Modelsim SE V.6.4b by Mentor Graphics, while fault coverage was computed using Synopsys Tetramax V. B-2008.09-SP3. Both programs run on an Intel Xeon E5450 @3.0 GHz with 8 GB RAM.

3.1 DMA Controller

The first benchmark SoC contains an 8051 compatible IP core (Oregano Systems, http://www.oreganosystems.at/), a 64 KB data RAM block, a 64 KB program ROM, an 8237-compliant DMA controller, and other peripherals with DMA transfer capability (see Fig. 3). A further logic module was designed and connected between the 8051 external memory port and the data bus of the DMAC, in order to setup the proper communication protocol and bus control signals. The module includes the required three-state buffers and filters the addresses generated by the processor core, relying on the fact that the employed processor outputs the address ‘00 h’ when not requiring the control of the data bus.

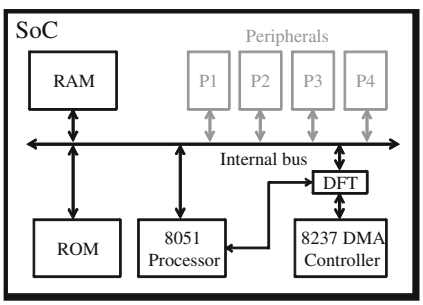

Fig. 3 Benchmark system for the DMA example. The DFT module is intended to facilitate the control of some DMA signals and data using the 8051 processor ports in order to emulate a peripheral behavior during test

The employed 8237-compliant controller owns four independently programmable transfer channels, and DMA requests can be activated via hardware or software. Memory-to-memory and peripheral-to-memory data transfers and block memory initialization are supported. Additionally, this DMAC offers static read/write or handshake modes, and includes direct bit set/reset capabilities. The DMAC is described in about 3,600 VHDL lines. After synthesis in an in-house developed standard cell library, the equivalent gate count is 10,227 and the number of stuck-at faults is 21,388.

Exploiting the described methodology, a compact initial test set for the DMA controller has been generated. Firstly configuration registers, such as the command register and mode register, are identified; secondly, coverage metrics are chosen. For the available DMAC model, the most important metrics are statement, condition and branch coverage.

Consequently, the resources within the controller are labeled as shared or unshared. The main shared resource is the DMA engine, followed by the general registers, the internal buses and the priority arbiter. The unshared resources are mainly related to channel control, such as address registers, word count registers and channel mode registers.

By discarding the invalid configuration words, we obtain a configuration graph composed of 56 nodes (Fig. 4). The DMAC priority logic performs a data transfer control which is not directly related to a specific channel, neither to an address increment or decrement policy. Thus, only one among the other resources, e.g. the DMA channels, is associated to each transfer mode of the DMAC. This is directly mapped on the configuration graph.

Exploiting a pseudo-random algorithm for visiting the graph, 5 configuration paths are obtained. In order to generate the actual test programs, different DMA transfers are programmed, involving memory and I/O peripherals. It is worthwhile noting that the effect of the DMA auto-initialization function, which involves a continuous data transfer until it is interrupted, is also considered: since this option implies the completion of a programmed transfer sequence, it is used with short data counts. Additionally, to completely excite the priority logic, the transfer request signals from each channel are activated in an exhaustive sequence of combinations.

By applying the whole method a first test bench is obtained aiming at saturating high-level metrics, counting about 1,000 assembly code lines and needing 6,658 clock cycles to be completed. Table 1 reports the coverage results obtained by the application of the complete test bench: all chosen metrics were completely saturated without the need of iterating the process. Some of the reported percentages do not reach 100% only because of dead code blocks, for example unreachable conditional branches.

Stuck-at fault coverage of the developed program set is 71.34%. This low value stems from the fact that this test bench has been generated starting from the high-level description,

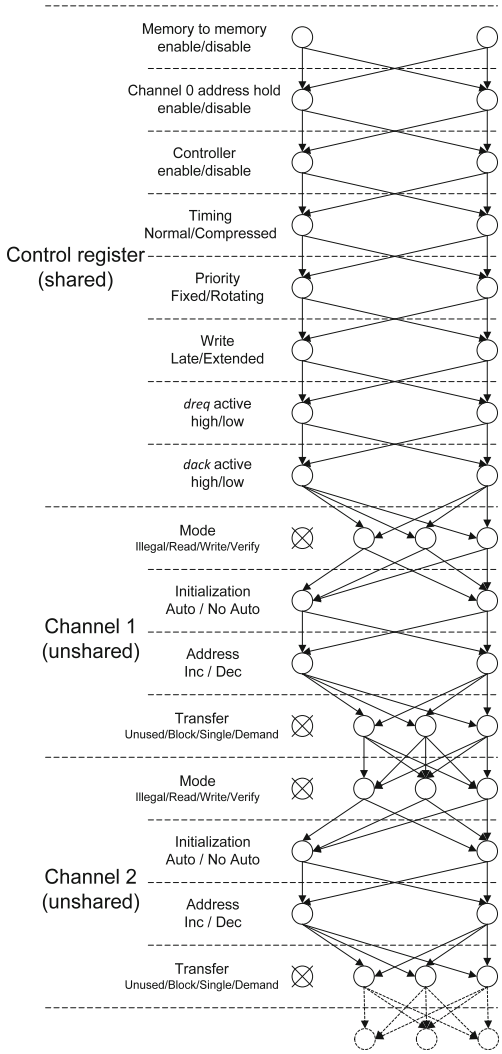

Fig. 4 Part of a configuration graph, representing the modules of an 8237 compliant DMAC

only. By following the additional test-related step, we were able to increase controllability and observability.

Table 1 Coverage results for the DMAC

	Total	Covered	
		#	%
Statements	2,144	2,093	97.62
Branches	628	605	96.34
Conditions	180	162	90.00
Fec Conditions	250	219	87.60
Expressions	7	7	100.00
Fec Expressions	10	10	100.00
Toggle Nodes	746	730	97.86
Stuck-at faults	21,388	15,258	71.34
Stuck-at faults (final)	21,388	20,419	95.50

Firstly, the test bench does not excite the unshared modules described by the same code lines. Therefore, the routines related to the unshared modules are replicated on all their instances.

Secondly, to saturate high-level metrics, the functionalities of the DMAC have to be stimulated but not necessarily observed from the outside. Conversely, during testing the fault effects have to be propagated to observable points, such as the processor bus. For this reason, register read procedures are added in relevant points in the available fragments.

Thirdly, the composing modules, such as counters, registers and encoders need to be stimulated according to their specific architectural features, following the guidelines reported in subsection 2.4.

The last row in Table 1 reports the stuck-at coverage of the final test program set, which counts about 1.7 K lines of assembly code and takes 15,371 clock cycles to complete. The stuck-at fault simulation takes about 170 s.

The whole test bench generation process performed by a skilled test engineer took about two weeks, considering the graph construction and analysis and the development of the test programs.

3.2 Interrupt Controller

The second embedded system used to prove the effectiveness of the proposed approach is based on the AMBA2.0 bus architecture and it is implemented employing the LEON3 grlib library [10]. It includes a LEON3 core with a set of system and I/O peripherals, and a SRAM memory core (Fig. 5). The verification target is the grlib interrupt controller (IRQmp), which is described by 340 VHDL code lines and can be employed in multiprocessor systems as well as in monoprocessor ones. It manages interrupt request levels, forwarding them to the CPU core(s). Level 15 has maximum priority and through the interrupt level register it is possible to assign an additional priority level (0 or 1, the latter one owning highest priority) for each of the 16 levels. After synthesis, the targeted module for a single-core implementation counts 1,923 equivalent gates and 4,413 stuck-at faults.

The priority encoder it the most important shared module and can be configured by writing the interrupt level register. This register provides the current global configuration of the system: in fact, its value indicates the order in which requests will be forwarded to the CPU. Conversely, each interrupt line is regarded as unshared and its configuration register is composed by a set of three bits resulting from pending register, force register and mask register. These three bits provide the configuration of each unshared module.

The obtained configuration graph is reported in Fig. 6 and features 120 nodes.

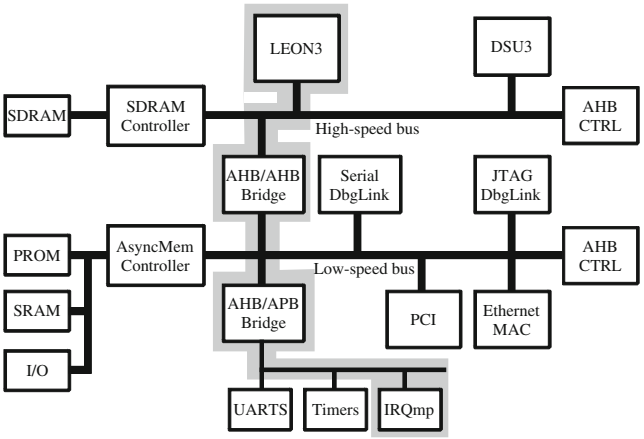

Fig. 5 Leon3 System with Interrupt Controller IRQmp. The part highlighted in gray shows the necessary path to access to the Interrupt Controller from the Leon 3 processor. Here we assume that an ATE supports the test application by suitably controlling the IRQmp inputs not directly controlled by the microprocessor

Only two configurations are used for generating the initial test set targeting the saturation of high-level metrics, which is composed of 232 assembly code lines and is executed by LEON3 in 5,300 clock cycles. Table 2 reports high-level and fault coverage figures obtained by this first test set.

The high-level coverage metrics figures are obtained without any iteration of the generation process, and they are saturated because the original description contains unreachable code lines. This means that the statements and conditions described in those lines can be activated only with specific values of some hardware configuration parameters (namely: *ncpu* and *eirq*, setting the number of cores and enabling extended interrupts, respectively) that are different from the ones set in the selected system implementation.

To improve fault coverage, additional tests have then been developed, by increasing the number of configurations in the unshared modules according to the guidelines in subsection 2.4. To enhance fault coverage in priority logic, a sequence of simultaneous interrupt requests has been developed, following an ATPG-derived test sequence for the priority encoder; simple interrupt services routines reading and writing memory checkpoint variables were added to monitor the resource behavior. The test program set counts 380 assembly code lines and requires 11,350 clock cycles for its execution. The last row in Table 2 reports the obtained stuck-at coverage value. The analysis and generation process took about six days, while the complete fault simulation took about 10 s.

For the sake of comparison, scan chain insertion has been performed into the interrupt controller circuit, obtaining a single 65 flip-flop chain. Automatic pattern generation provides 134 patterns reaching 100.0% stuck-at fault coverage. Considering a circuit running frequency of 200 MHz and an ATE scan chain upload/download frequency of 50 MHz, the developed functional test takes about 0.057 ms, while the scan-based one requires 0.18 ms.

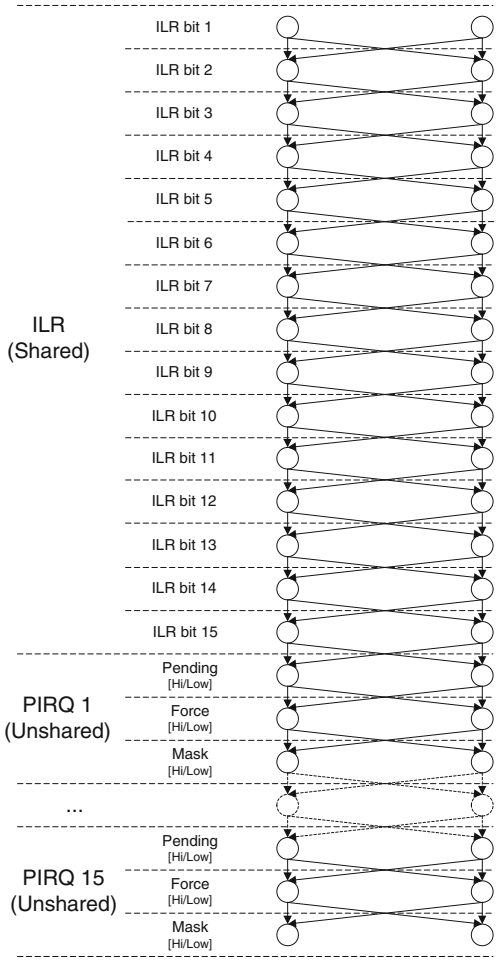

Fig. 6 Part of a configuration graph representing all the possible configurations for the IRQmp

4 Conclusion

In this paper, we presented a structured methodology for testing system peripherals in a functional manner; the methodology can be used by test engineers to tackle the test issues in a results-oriented and effective way. The described

Table 2 Coverage results for the IRQ controller

	Total	Covered	
		#	%
Statements	114	82	71.93
Branches	85	56	65.88
Conditions	4	4	100.00
Stuck-at faults	4,413	4,231	95.87
Stuck-at faults (final)	4,413	4,357	98.73

methodology is based on the SBST paradigm that exploits a set of assembly programs executed by the system processor in order to apply the testing procedures. To produce a compact and efficient test set, a minimal set of configuration paths are derived from the configuration graph. Additionally, the proposed methodology also includes some guidelines to support the generation of test programs for some of the most common modules that belong to system peripherals.

The reported examples experimentally demonstrate the method effectiveness and flexibility when tackling different system peripherals. In fact, the resulting test sets reached high coverage figures while keeping low memory occupation as well as testing execution time. Finally, considering the functional characteristics of the resulting test sets, it is possible to apply the obtained stimuli sets for manufacturing test at the end of the production line, as well as for incoming inspection.

The high stuck-at fault coverage obtained, together with the at-speed test application, guarantees a satisfactory defect coverage. Works are ongoing to extend the proposed technique to improve dynamic fault coverage as well, through the integration of structural circuit information into the flow.

Acknowledgment The authors would like to thank Danilo Ravotto and Alberto Giraudo for the useful discussions and for performing most of the experiments.

References

1. Apostolakis A, Psarakis M, Gizopoulos D, Paschalis A (2007) Functional processor-based testing of communication peripherals in systems-on-chip. IEEE Trans VLSI Syst 15(8):971–975
2. Apostolakis A, Gizopoulos D, Psarakis M, Ravotto D, Sonza Reorda M (2009) Test program generation for communication peripherals in processor-based SoC devices. IEEE Des Test Comput 26(2):52–63
3. Bolzani L, Sanchez E, Schillaci M, Sonza Reorda M, Squillero G (2007) An automated methodology for cogeneration of test blocks for peripheral cores. IEEE Int'l On-Line Testing Symposium, pp 265–270
4. Bushnell ML, Agrawal VD (2000) Essential of electronic testing. Springer
5. Chandramouli R, Pateras S (1996) Testing systems on a chip. IEEE Spectrum, pp 1081–1093
6. Chen L, Ravi S, Raghunathan A, Dey S (2003) A scalable software-based self-test methodology for programmable processors. In: Proc. IEEE/ACM Design Automation Conf. 548–553
7. Corno F, Sanchez E, Sonza Reorda M, Squillero G (2004) Automatic test program generation – a case study. IEEE Des Test Comput 21 (2):102–109
8. Dushina J, Benjamin M, Geist D (2003) Semi-formal test generation and resolving a temporal abstraction problem in practice: industrial application. IEEE/ACM Design Automation Conference, pp 699–704.
9. Gizopoulos D, Psarakis M, Hatzimihail M, Maniatakos M, Paschalis A, Raghunathan A, Ravi S (2008) Systematic software-based self-test for pipelined processors. IEEE Trans VLSI Syst 16(11):1441–1453
10. GRLIB IP Library 1.1.0, Aeroflex Gaisler, www.gaisler.com/cms/
11. Grosso M, Pérez Holguin WJ, Ravotto D, Sanchez E, Sonza Reorda M, Velasco Medina J (2010) Functional test generation for DMA controllers. 11th IEEE Latin-American Test Workshop, pp 1–6.
12. Grosso M, Pérez Holguin WJ, Ravotto D, Sanchez E, Sonza Reorda M, Velasco Medina J (2010) A software-based self-test methodology for system peripherals. 15th IEEE European Test Symposium pp 195–200.
13. Huang J-R, Iyer MK, Cheng K-T (2001) A self-test methodology for IP cores in bus-based programmable SoCs. IEEE VLSI Test Symposium, pp 198–203.
14. Hwang S, Abraham JA (2001) Reuse of addressable system bus for SoC testing. IEEE Int'l ASIC/SoC Conference, pp 215–219.
15. Kranitis N, Paschalis A, Gizopoulos D, Xenoulis G (2005) Software-based self-testing of embedded processors. IEEE Trans Comput 54(4):461–475
16. Krstic A, Chen L, Lai W-C, Cheng K-T, Dey S (2002) Embedded software-based self-test for programmable core-based designs. IEEE Des Test Comput 19(4):18–27
17. Lai W-C, Cheng K-T (2001) Instruction-level DFT for testing processor and IP cores in system-on-a-chip. IEEE/ACM Design Automation Conference, pp 59–64.
18. Lingappan L, Jha NK (2007) Satisfiability-based automatic test program generation and design for testability for microprocessors. IEEE T VLSI SYST 15(5):518–530
19. Maxwell PC, Aitken RC, Johansen V, Chiang I (1991) The effect of different test sets on quality level prediction: when is 80% better than 90%? IEEE Int Test Conf pp 358–364
20. Parvathala P, Maneparambil K, Lindsay W (2002) FRITS – a microprocessor functional BIST method. IEEE Int Test Conf 590–598
21. Psarakis M, Gizopoulos D, Sanchez E, Sonza Reorda M (2010) Microprocessor software-based self-testing. IEEE Des Test Comput 27(3):4–19
22. Shen J, Abraham J (1998) Native mode functional test generation for processors with applications to self-test and design validation. In: Proc. IEEE Int Test Conf 990–999
23. Thatte S, Abraham J (1980) Test generation for microprocessors. IEEE Trans Comput 29(6):429–441

Michelangelo Grosso is a postdoctoral fellow at the Department of Control and Computer Engineering of Politecnico di Torino (Torino, Italy). He received the MS degree in Electronic Engineering in 2004 and the PhD degree in Computers and Systems Engineering, in 2008, both from Politecnico di Torino. His research interests range from verification and test to reliability of integrated circuits and systems.

Wilson Javier Perez Holguin has received a MSc from Universidad-Nacional de Colombia and currently he is a PhD candidate at Universidad del Valle, Colombia. He is an assistant professor at UPTC, Sogamoso, Colombia. His research areas include Design&Testing of Digital Electronic Circuits and Industrial Automation.

Ernesto Sanchez received his degree in Electronic Engineering from Universidad Javeriana - Bogota, Colombia in 2000. In 2006, he received his Ph.D. degree in Computer Engineering from the Politecnico di Torino, where currently, he is an Assistant Professor with Dipartimento di Automatica e Informatica. His main research interests include microprocessor testing and Evolutionary Algorithms.

Matteo Sonza Reorda is a full professor in the Department of Control and Computer Engineering at Politecnico di Torino. His research interests include testing and fault-tolerant design of electronic circuits and systems. He has a PhD in computer engineering from Politecnico di Torino. He is a senior member of the IEEE. He serves on the editorial board of the Journal of Electronic Testing: Theory and Applications.

Alberto Paolo Tonda received his M.S. degree in computer science engineering in 2007 from Politecnico di Torino, Torino, Italy, and is currently a Ph.D. Student of computer science engineering at the same institution. His research interests include verification of electronic systems and application of evolutionary techniques to industrial problems.

Jaime Velasco-Medina received the PhD and MSc degree from TIMA/INPG, France. Currently, he is professor of the School of Electrical and Electronics Engineering at Universidad del Valle in Cali, Colombia. His research interest are mixed-signal circuits test, digital systems design and bionanoelectronics.