Functional Verification of DMA Controllers

Functional Verification of DMA Controllers

Michelangelo Grosso · Wilson Javier Perez Holguin · Danilo Ravotto · Ernesto Sanchez · Matteo Sonza Reorda · Alberto Tonda · Jaime Velasco Medina

Abstract Today's SoCs are composed of a wide variety of modules, such as microprocessor cores, memories, peripherals, and customized blocks directly related to the targeted application. To effectively perform simulation-based design verification of peripheral cores, it is necessary to stimulate the description in a broad range of behavior possibilities, checking the produced results. Different strategies for generating suitable stimuli have been proposed by the research community to functionally verify these modules and their interconnection when embedded in a SoC: however, their verification often remains a largely manual and unstructured operation. In this paper we describe a general approach to develop concise and effective sets of inputs by modeling the configuration modes of a peripheral with a graph, and creating paths able to cover all of its nodes: proper stimuli for the device are then directly derived from the paths. The resulting inputs sequences are aimed at design verification of system peripherals such as DMA controllers, and can be applied via simulation by means of dedicated testbenches or by setting up an environment including a processor, which executes a proper test priogram. In the latter case, the developed programs can be exploited in later stages for testing, by adding suitable observability features. Experimental results demonstrating the method effectiveness are reported.

Keywords Design verification · Test program · Set stimuli generation · DMA Controller

Responsible Editor: F. Vargas

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

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

D. Ravotto
e-mail: danilo.ravotto@polito.it

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

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

W. J. Perez Holguin · J. V. Medina
Bionanoelectronics Group, Universidad del Valle Cali, Cali, Colombia

W. J. Perez Holguin
e-mail: wjperezh@univalle.edu.co

W. J. Perez Holguin
e-mail: wilson.perez@uptc.edu.co

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

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

1 Introduction

Most Systems on Chip (SoCs) integrate at least one processor core, some peripheral devices, different logic modules, and a variable number of memory cores. Although the SoC design paradigm may simplify the design phase, it also increases the complexity of the validation, verification and testing (VV&T) process, by combining modules from different sources, design styles, and test characteristics.

Until now, the research community has devoted considerable efforts to VV&T processes for processors, while integrated peripherals have been less investigated. Naturally, processor cores have focused the research attention due to their area occupation and their primary role in computation and management tasks within the SoC. However, it must be noted that the increasing number and complexity of embedded peripherals strongly characterizes most of the current SoCs.

In general, design verification is the process of verifying that all modeled behaviors of a design are consistent with a reference model. The reference model may represent a set of properties that the system needs to fulfill, and usually it is described at a higher abstraction level [14]. In synthesis, verification is defined as the process that aims at guaranteeing the correct translation of the model delivered at a certain abstraction level to its successive, less abstract, model.

Design verification methodologies have been developed in a broad spectrum, ranging from manual verification to formal verification techniques, and including, for example, random and semi-random approaches. Formal verification uses mathematical techniques to prove the correctness of the design, but it frequently involves counting on enormous computational resources [8], even for some simplified models. On the other hand, despite the simulation-based methods can never guarantee the complete conformance to specification of formal methods, they are widely used due to the reasonable amount of computational resources required, the grade of details of the circuit behavior that can be simulated and their potential to detect and diagnose design and implementation errors in different stages of the device VV&T processes [8, 16].

In the simulation-based context, it is possible to state that a verifying strategy implies to follow a simulation-based procedure in which input data (called *set of stimuli*) are applied to a model of the device under evaluation (called *device under test* or DUT). Subsequently, the observed and expected behaviors are compared by means of a *response checker* that generates pass/fail information regarding the outcome of the comparison [16].

Simulation-based methodologies aim at uncovering design errors by thoroughly exciting the current model of the circuit using suitable sets of stimuli [16], which can be randomly, manually or automatically generated. In any case, the objective is to fully excite all functions of the DUT by employing a reduced number of appropriate stimuli. This issue is a crucial point of the whole methodology, since it strongly affects the cost of the whole VV&T process, and because the simulation of an exhaustive set of stimuli is generally far too expensive [8]. However, determining the appropriate stimuli set is a problem-dependent and non-trivial task.

In order to quantify the verification progress, it is possible to determine the quality of the stimuli set by measuring the *coverage* obtained by the stimuli on the DUT. Usually, code coverage metrics, as detailed in [20], represent a good indicator of the stimuli set goodness. These metrics [13] can be classified as *implicit*, if inherent in the representation of the abstraction level from which the metric is taken, as in Register-Transfer Level (RTL) code coverage, or *explicit*, if defined by the verification engineer. Another classification can be introduced on the metric source, which can relate to implementation or specification.

Once a suitable set of stimuli has been obtained, the test set can be applied to the device through a *testbench*: this term usually refers to simulation code used to feed a predetermined input sequence to a design and to observe the response [4]. A testbench is commonly described using some Hardware Description Language (HDL) or Hardware Verification Language (HVL), but it may also include external data files or routines.

When dealing with peripherals designed to heavily interact with a processor, a common practice consists in creating a simulation environment modeling a small system, often including a processor and a memory, besides the target peripheral: in this case the testbench includes specially crafted programs that, when executed by the processor, produce the desired stimuli at the inputs of the DUT. In this way the readability of the testbench is increased, while the cost for creating and debugging it is reduced, since it is not necessary to create from scratch a dedicated testbench able to handle communication protocols. Additionally, when peripherals are included into a SoC, the correct connection and interaction between different modules must be verified as well; thus, some parts of the mentioned test programs could also be re-used to cope with these purposes. This approach shares many similarities with the technique known as *Software-Based Self-Test* or SBST [15], whose key idea is to exploit on-chip programmable resources to run test programs that suitably stimulate the processor itself and/or other devices accessible by it. A verification program for system peripherals may also constitute an effective starting point for developing a manufacturing test suite in later development phases: as a matter of fact, testbench-like stimuli cannot in general be easily applied to the input/output ports of a deeply embedded core.

This paper aims at providing verification engineers with a structured approach to generate stimuli for the verification of DMA controllers. This type of peripherals, called *system peripherals*, are in charge of providing system services, in contrast with peripherals devoted to communications services, called *I/O peripherals*. The group also includes the interrupt controllers, and timers, for example.

In particular, this paper presents a method for the generation of suitable stimuli able to thoroughly excite the different functionalities implemented by a typical DMA controller, starting from its functional description, only. The effectiveness of the generated stimuli can be evaluated using code or functional coverage metrics on the available device description. The resulting stimuli can be effectively used for the validation and verification of the core design, and later, they may represent an effective starting point for the development of a manufacturing test set. When compared to other approaches like [18], it is interesting to notice that the proposed methodology does not need a HDL description of the device or an abstraction procedure, which may cause loss of information. Additionally, the proposed methodology performs better than [11] due to improvements in one of the involved algorithms.

The rest of the paper is organized as follows: Section 2 provides some significant background material in the area of peripheral verification, as well as on the metrics used for evaluating the effectiveness of generated stimuli, and outlines the main features of typical DMA controllers. Section 3 describes the method that we propose for generating proper stimuli for DMA controller verification. A case study is presented in Section 4, whereas Section 5 reports some preliminary experimental results gathered on a representative test case. Finally, Section 6 draws some conclusions, and summarizes future research activities in the area.

2 Peripheral Verification and Test

2.1 Background

As the design complexity of single cores and whole SoCs increases, activities of logic inspection and verification are becoming more and more difficult, especially the latter, which is often regarded as a major bottleneck in the whole design cycle [21]. Because of the broad application of Intellectual Property (IP), the increasing work efficiency of IP core simulation and verification will obviously reduce design risk and complexity. In this context, functional verification is applied to determine whether the design respects the initial specifications.

Functional verification can be accomplished using three complementary approaches: black-box, white-box, and grey-box methods [4]. Black-box verification does not depend on the specific implementation of the DUT, but it makes it difficult to control and observe specific features. A white-box approach has full visibility and controllability of the internal structure of the design being verified: it lets verification engineers configure test inputs rapidly and efficiently isolate functions, but it has poor portability and the design needs to be described in great detail. Grey-box is a hybrid method which controls and observes a design through its top-level interfaces, but it is meant to excite implementation-specific features.

In order to precisely excite parts of a SoC, suitable and compact stimuli sets must be devised. In literature, several approaches have been proposed to reach this objective. In [17], for example, Rosenberg presents a manual stimuli generation technique based on coverage analysis, where the testbench completeness is achieved by the identification of the less stimulated coverage aspects and, then, indicating the manner of changing the stimuli generation pattern. An automated approach is presented by Bose et al. in [6], where a genetic algorithm automatically generates biases which are then used for a biased random instruction generator applied to a PowerPC. Braun et al. show that even Bayesian networks and data mining techniques can be used to improve direct random simulation with an automatic feedback loop [7], reaching interesting results.

A suitable set of stimuli needs not only to be comprehensive enough, but its execution length has to be taken into account as well. In [19], Strum et al. compare two testbench techniques, and use filtering of redundant stimuli to reduce testbench execution times.

It is interesting to note that most of the available works developed by the research community on VV&T issues mainly tackle processor cores. These methodologies often resort to functional approaches based on exercising specific functions and resources of the processor [18]. Verification stimuli generation for embedded processors is also tackled by SBST techniques, where the processor is stimulated by making itself run specifically generated verification programs. In [15] Psarakis et al. present a comprehensive study about the potential role of software-based self-testing in the microprocessor VV&T process, as well as its supplementary role in other classic functional-and structural-test methods. Additionally, the authors propose a well structured taxonomy for different SBST methodologies according to their test program develop-ment philosophy.

Concerning system peripherals, Dushina et al. [18] introduce a semi-formal based methodology to generate test sets targeting corner cases of the device under test. The device is modeled in a simplified manner and translated into a Finite State Machine (FSM). Each FSM state corresponds to a combination of the coverage variables. Then, a set of abstract tests containing a sequence of states for every case is obtained via a coverage-driven test generator. Finally, every single abstract sequence of states is translated to the real test set. This methodology was used for the verification of a DMA controller embedded in a RISC-based microcontroller, achieving about 87% of statement coverage and 75% of branch coverage.

Two newer approaches were presented in [11] and [12] by Grosso et al. aimed to system peripherals verification and test. These works propose a method to develop functional tests for DMA controllers, and form the basis for the approach presented herein. Some preliminary experimental results for an 8237-compliant DMA controller core were reported to demonstrate the method effectiveness.

Other approaches can be found in literature related to VV&T issues for I/O peripherals. For example, Bolzani et al. in [5] propose a fully automated methodology for the generation of test programs for peripheral cores (PIA, UART) embedded in a SoC. The methodology is based on the exploitation of the correlation between high-level metrics (*Toggle*, *Expression*, *Condition*, *Branch* and *Statement*) and the gate-level fault coverage. Apostolakis et al. [3] present a generic deterministic flow for the application of processor-based testing to communication peripheral cores. In this approach, the test sets for the individual subcomponents of the communication peripheral core are pre-computed and pre-generated. In [2] the authors propose a combination of the approaches presented in [5] and [3] and define a new automatic methodology that has been evaluated on a SoC with three popular communication peripherals such as UART, HDLC and Ethernet. In these cases, the verification stimuli are often applied by simulating a complete system including the DUT.

2.2 Test Generation and Coverage Metrics

Coverage metrics were firstly defined in software testing as the measure of how thoroughly exercised a given piece of code is by a test set; this goal can be achieved by quantifying the capacity of a given set of input stimuli to activate specific features of the model [10]. Similarly, borrowing the idea from software testing [20], it is possible to state that the adequacy of a set of stimuli can be measured using well defined coverage metrics (e.g., the percentage of statements in the circuit description which are activated during the testbench simulation, or *statement coverage*) when dealing with digital circuit validation, verification or testing procedures. In addition, if the coverage metrics are employed in a generation process, the collected information can be exploited as a useful test criterion [14].

The selection of the most suitable coverage metric to be used for evaluating the outcome of the generation process depends on the characteristics of the considered model. Both code coverage and functional metrics, defined below, can be used for this purpose:

- *Code coverage metrics* derive directly from metrics used in software testing. These metrics identify which code structures belonging to the circuit design described in HDL are exercised by the set of stimuli, and whether the control flow graph corresponding to the code description is thoroughly traversed. The structures exploited by code coverage metrics range from a single line of code to if-then-else constructs.
- *Functional coverage metrics* target design functionalities during the validation, verification and test processes. These metrics correspond to a set of specific operative modes that exercise the design in well defined or restricted situations, guaranteeing that the design under evaluation complies with the design functionalities as described in the specifications. Often, the functionalities to be covered are summarized in a check table.

2.3 Direct Memory Access Controller Description

A Direct Memory Access (DMA) Controller, or DMAC, is designed to allow large blocks of data being transferred between memory and peripherals (or between two memories) without the intervention of the microprocessor. Once the DMA registers are programmed by the processor, a transfer can be started in order to either relocate data from a memory location to another or write data to/from a peripheral depending on the application requirements. The inclusion of a DMAC into a system reduces the microprocessor workload, since different devices are able to transfer data without requiring the microprocessor's intervention. During data transfers, the microprocessor is allowed to perform other tasks; sometimes, the data transfer is also faster when executed by a DMAC.

Although their size is normally not huge, DMACs may exhibit a significant complexity. Features supported by DMACs include, among others: different data transfer modes, management of several peripheral channels, programmable arbitration mechanisms to manage multiple data transfers. In fact, the latter point is particularly difficult to handle, since it requires taking into account several specific situations, coming from different arbitration mechanisms and different sequences of DMA requests from peripherals.

Figure 1 sketches the architecture of a generic DMAC. Roughly speaking, a multichannel DMAC is composed of a set of configuration and status registers, a set of transfer channels, a priority arbiter, and a DMA engine which includes a state machine and a logic block. The configuration registers provide the whole device with the operation settings, including the necessary information to properly perform all the DMA operations. Configuration and status registers can be accessed by the processor through a system bus. Depending on the application, it is possible to activate a microprocessor interrupt at the end of a DMA channel transfer.

Every single channel counts on a particular set of registers that allow the channel to perform independent data transfers. The elements involved in a data transfer are:

r

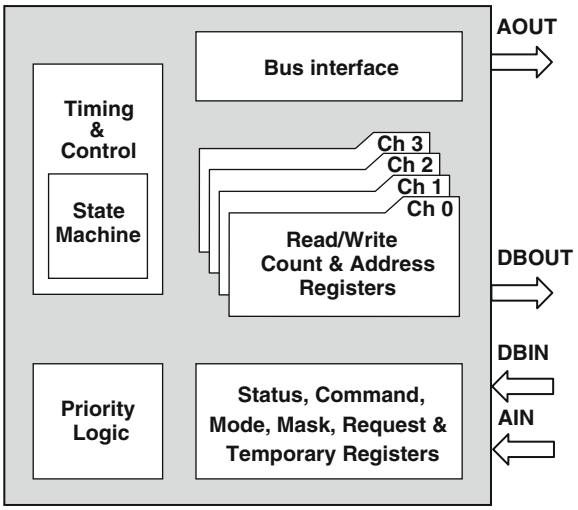

Fig. 1 DMAC generic architecture

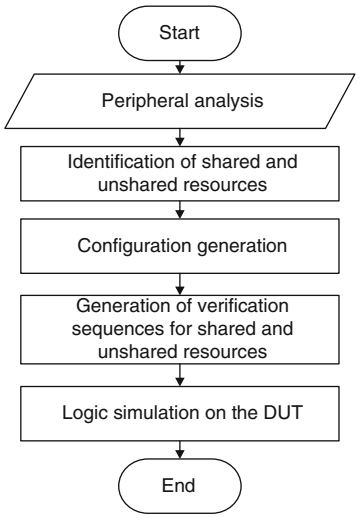

Fig. 2 Proposed methodology flow description

a source address, a destination address, the number and dimension of the data words to transfer, and finally, the transmission mode selected.

Making use of the DMA arbiter it is possible to implement different priority modes when different transfers are required at the same time: two of the most commonly adopted are fixed priority and round robin.

3 Proposed Approach

When tackling embedded system peripherals, devising a stimuli generation technique based on a purely functional approach is a non-trivial task. For instance, setting every channel of a DMAC and checking its correct behavior in all possible configurations would create a set of stimuli of prohibitive dimensions. Furthermore, once the device is configured, there is also a large number of possibilities for the data and the dimension of the blocks to be transmitted. While a good verification set for the device must be able to thoroughly excite it, it should also contain a limited number of configuration sets in order to be applicable in real scenarios.

We propose a structured methodology for the development of an effective and compact verification stimuli set for system peripherals, focusing our attention mainly on DMA controllers, based on the analysis of a high-level description of the addressed devices, and providing an operative strategy to verification engineers. Figure 2 reports a schema of the approach that aims at reducing the complexity of the produced verification set, guaranteeing at the same time its ability to thoroughly excite the DUT. Coverage metrics are collected at the end of the stimuli generation process, and these are employed to assess the efficacy of the generated stimuli set.

The first step of this methodology concerns the analysis of the description of the peripheral core, and has two main objectives: firstly, it allows to identify the main elements devoted to the configuration of the device, e.g., the configuration registers; and secondly, it leads to identify, depending on the available description of the DUT, the most suitable metrics to evaluate the completeness of the generated stimuli set.

The second step is devoted to identify *shared* and *unshared* resources within the peripheral under test. A shared resource corresponds to some circuitry which is exploited during the management and usage of different channels, while an unshared one participates in the elaboration related to a single channel, only. A *shared* resource can be either *control shared* or *elaboration shared*:

- *Control shared* resources are mainly devoted to drive and coordinate the operation of internal peripheral modules;
- *Elaboration shared* resources perform specific tasks during data elaboration, and are unique along the data path. These modules may receive or send data to one of a set of internal sub modules (*unshared*) that implement an identical function.

Unshared resources, on the other hand, are mainly involved in the data path as elaboration resources, and are usually present as a set of different modules able to carry out the same function described by the same HDL code, often in order to introduce parallelism.

The labeling of resources as shared or unshared can be easily performed on the basis of a high-level description of the DUT. In Fig. 1, for example, the DMA *timing & control* module would be labeled as *elaboration shared* because every channel uses functionalities performed by it to complete data transfers, while, on the other hand, the

priority arbiter would be labeled as *control shared*, for its control functions. In any case, it is enough to label both modules as shared in order to complete the next steps of the methodology. All channel modules, on the other hand, are clearly *unshared*.

In the third step, all possible configurations of the DUT are subsequently represented by a *configuration graph*: the main goal of this graph is to provide the methodology with a well organized structure that allows reducing the number of configurations of the device whereas avoiding unnecessary replications. This step provides the user with a reduced set of device configurations that is intended to support the creation of verification stimuli sequences. The actual stimuli generation is performed in the fourth step, considering for every sequence the whole device configuration, as well as special requirements for each configuration path.

Conceptually, the configuration graph is a directed graph, whose nodes are grouped in levels, where all nodes in the same level correspond to the same group of configuration bits that control a certain functional characteristic of some device resource; each node in a group represents a value that the group can assume. An arc in the graph from node A to node B that 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. Several consecutive levels represent all possibilities in the configuration register of a given resource. It is important to notice that a path in the configuration graph can touch only one node in each level, since nodes in each level represent mutually exclusive configurations of a specific device resource. A valid path in the graph starts at the first level and ends in the final one, and represents a legal configuration setting for the device. Figure 3 reports a schematic view of a configuration graph of a shared module and two unshared modules. Nodes in each level represent alternative settings of configuration bits driving the same functionality: for example SM [14, 16] together set the device in three different modes, because one of their possible configurations is prohibited. Grey arrows show all possible arcs in the graph, while black arrows identify a specific configuration path for the device.

Reducing the number of considered paths is a priority objective during stimuli generation, to avoid the explosion of the number of configurations needed and consequently the size of the stimuli set.

A first reduction of the configuration graph is performed exploiting information gathered in the previous steps. The maximum number of different configurations available for the device can be determined by analyzing the configuration registers: usually, the upper bound consists of 2^n possibilities, where n is the number of bits contained in all the configuration registers. Not all the 2^n configurations,

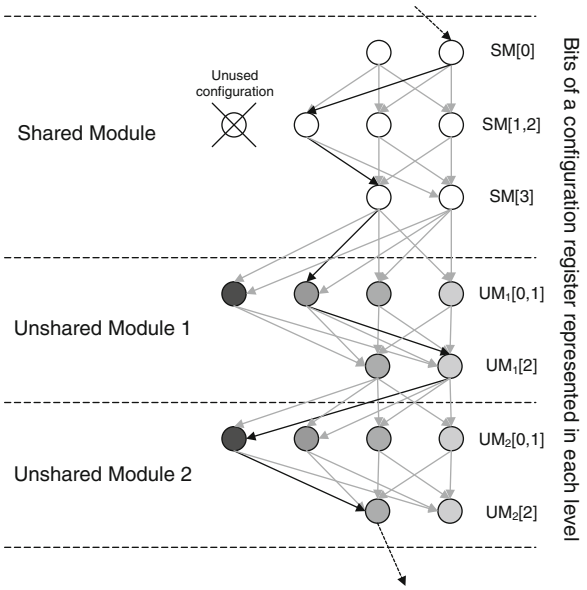

Fig. 3 A sample path in a configuration graph. On the right, the bits of the configuration register represented in each level are reported. Grey arrows depict all the possible paths/configurations, while black arrows illustrate a sample path/configuration. Corresponding nodes in unshared modules are shown with the same shades of grey

though, are actually useful or valid, and in some cases there are incompatibilities to consider. Impossible configurations do not appear in the configuration graph of the device, but this reduction alone is usually not significant.

Labeling modules as shared and unshared is then used to further reduce the number of paths needed to cover the entire configuration graph: since sets of unshared modules perform the same functions in the DUT, it is not necessary to excite all configurations of all the shared and unshared modules in a set with regards to the objective of maximizing code coverage.

Theoretically, to verify all combinations of the DUT configurations, at least each arc of the graph should be covered by one path; experimental evidence, however, suggests that it is sufficient to cover all nodes to obtain a satisfactory coverage on the chosen metrics guaranteeing the stimuli set capacity, as it will be shown in Section 5. This constraint alone considerably reduces the number of paths needed to thoroughly excite the device. An intuitive reason to justify experimental data could be the following: consider two independent options with two possible settings each, for example priority (which can be either fixed or rotating) and timing (which can be either normal or compressed). Covering the nodes corresponding to each setting generates two paths, while covering the arcs between them creates four: since the two options are independent, the latter approach originates two more paths that do not provide any coverage information with respect to the first two.

Let us, for example, refer to Fig. 1: as stated above, the priority logic is labeled as shared, while channel modules are unshared. To exhaustively excite the DUT, it is important to stimulate all configurations of the DMA engine, but is not necessary to use every channel in every priority logic configuration.

Once the aforementioned operations are carried out, a list of configuration paths must be obtained, following a suitable algorithm to efficiently visit the nodes. The goal of the 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. In addition, there could be both *compulsory paths* specified by the verification engineers and *prohibited configurations* inherent to the DUT.

Compulsory paths represent specific corner cases or configurations with great relevance, which may correspond to specific operative modes defined in functional coverage metrics, and whose resulting behavior the verification engineers desire to check. Thus, they are always included in the list of paths, regardless of the algorithm used to visit the nodes. For example, while tackling a DMAC, a verification expert could be interested in performing a memory-to-memory transfer while enabling the channel 0 address hold.

Prohibited configurations are expressed as rules that prevent specific nodes in different levels from being part of the same path, and they must be taken into account each time the algorithm chooses a node included in one of those rules. For example, if the memory-to-memory option is enabled in the command register of a DMAC, the compressed timing option cannot be activated, since the memory bus cannot be used at the same time for reading and writing.

Figure 4 outlines the algorithm exploited to identify the set of configuration paths. In this algorithm, Pt is the set of configuration paths, whereas p is the path under construction.

Initially, all nodes in the graph are labeled as unvisited, and every arc is tagged with a 0 weight. Compulsory paths are firstly identified (line 1) by making use of graph information gathered before; nodes belonging to them are marked as visited.

Then, if there are still unvisited nodes in the graph, for every available level in the graph, a node is selected, and

```
1 Pt ← compulsory_paths(graph);
2 If (unvisited_nodes)
3  p ← null;
4  foreach(level)
5   p ← choose_node(graph);
6  end;
7  Pt ← p;
8 endif;
```

Fig. 4 Paths compilation algorithm

included in the current path. In order to correctly select nodes that may be included in the current path p, the function choose_node(graph) (line 5) evaluates prohibited conditions, and updates nodes and arcs weights. If more than one node is available in each level, the function chooses one of the possibilities, taking into account graph constraints and graph information. When compared to [11], the current algorithm exploits weights associated to each node to visit the graph more efficiently, reducing the number of configuration paths needed to obtain better coverage figures.

In the fourth step, the actual test algorithms are defined. Such algorithms are based on the main functionalities of each module that have to be identified:

- Firstly, *shared* modules are analyzed. For example, the priority logic module is devoted to schedule different channels transfer activity following the configuration settings. However, as a *shared* device, the priority arbiter performs its handling activities on the complete set of DMA channels regardless of the specific mode of operation of every one;
- Secondly, *unshared* modules are considered. For example, the main activity of the hardware involved in every channel module of the DMAC is to count a series of words to transfer, in increasing or decreasing order, bounded by the limits previously defined in the channel internal registers. A counter can be thoroughly excited quite easily, making it operate in increasing and decreasing order alternatively. This directly descends from the paths derived in the previous step.

The verification algorithms defined for *unshared* modules may require specific restrictions regarding the configuration of the complete device: not all tests devised for an unshared module may be applied if a shared module forces it to operate in a specific mode. Usually, however, these test algorithms may be flexibly adapted to comply with different global device configurations, as in the case of the DMAC. Input stimuli sequences and expected results are hence developed. Each stimuli sequence originates from a path in the graph, and is composed of a configuration part and an operation part. The former directly descends from the configuration described in the path, and mainly consists in writing configuration registers of the addressed system peripheral. The latter part is aimed at activating the device under verification with a suitable set of data patterns, which must comply with the selected configuration and must be vast and varied enough to thoroughly excite the components. The operation part can include data to be preloaded in memory models, values to be written into registers and direct peripheral stimuli. Since each configuration is given in specific terms through the paths derived from the previous step, this task is

relatively straightforward and easy to implement, relying on the functional specification of the addressed device.

The peripheral stimuli are usually applied resorting to a dedicated HDL testbench that applies the required logic values to the input ports of the addressed module, emulating suitable protocols, and checks the values produced on the output ports. In this case the input stimuli sequence can be described using the constructs and abstractions of the selected HDL or VDL. Alternatively, programmable components belonging to a more complex simulation environment can be used to drive the peripheral ports similarly to what would happen during normal operations once the peripheral core is embedded in a system, at the expense of longer simulation time. In the case of the DMA controller, a segment code or test program can be written to be run by the system processor to interact with the addressed module and perform the sequence of operations as derived from the configurations and algorithms defined in the previous steps.

In the fifth step (Fig. 2), a logic simulation is performed: its aim is to assess the correctness of the generated stimuli sequences and evaluate the coverage metrics selected in the first step of the workflow. Code coverage figures are usually made available by commercial HDL simulation tools. Conversely, functional coverage metrics are usually evaluated manually: the verification engineer checks in the functional verification list if every one of the expected operative modes is activated by the obtained stimuli set. Interestingly, it is possible that a configuration path, derived from the configuration graph, covers more than one of the desired functionalities, for example when two channels are configured at the same time in two different operation modes.

At the end of the flow, if the results did not reach satisfactory levels, a new test case could be generated by taking into account previously discarded paths of the configuration graph or inserting new compulsory paths that cover the missing elements unveiled by coverage analysis. For example, when considering functional coverage metrics, a new compulsory path in the configuration graph can be set by the verification engineer to activate a specific operating mode. This need, however, never arose during the experiments described in the next sections.

4 Case Study

In order to assess its effectiveness, the proposed approach has been tested on the implementation of an 8237-compliant DMA controller. The addressed module provides four independently programmable transfer channels. DMA requests can be activated via hardware or software. The device allows to control memory-to-memory, memory-to-peripheral and peripheral-to-memory data transfers, and provides block memory initialization capability. Additionally, it offers static read/write or handshake modes, and includes direct bit set/reset capabilities.

Table 1 reports the main characteristics of the RTL description of the DMA controller. The configuration of the DMA controller channels is accomplished by setting each of the bits in the mode register, partially depicted in Fig. 5.

To effectively visit the configuration graph, a weighted pseudo-random algorithm is chosen to leverage non-deterministic selection of the paths, following the indications given in the previous section. Each node is associated with a weight which is initially set to 0 and incremented each time the node becomes part of a path. While creating a path in the configuration graph, a node will be visited if it has the lowest weight value among all nodes in the same level. If two or more nodes in the level share the same value, the node will be chosen randomly with equal probability among those with the lower weight value. The algorithm takes into account both compulsory paths and prohibited configurations.

The algorithm was implemented in about 350 lines of Java code, while the configuration files describing available configuration registers, compulsory path and prohibited configurations were written in extended markup language (XML) and, in this case, occupy about 4.73 KBytes of memory.

The stimuli application phase has been put into practice using two different approaches:

- In the first, a dedicated VHDL testbench has been developed to directly apply the required set of stimuli to the considered DMAC. The developed testbench, besides the DUT, includes the behavioral description of a memory core, and ad hoc functions that emulate the peripheral communications that would take place in a complete system.
- In the second, a verification environment has been setup, where the DMAC has been integrated into a simple microprocessor-based system, with the verification

Table 1 DMA controller description at RT-Level

DMAC	
Files	22
Instances	38
VHDL code lines	3,600
Statements	2,144
Branches	628
Conditions	180
Fec conditions	250
Expressions	7
Fec expressions	10
Toggle nodes	746

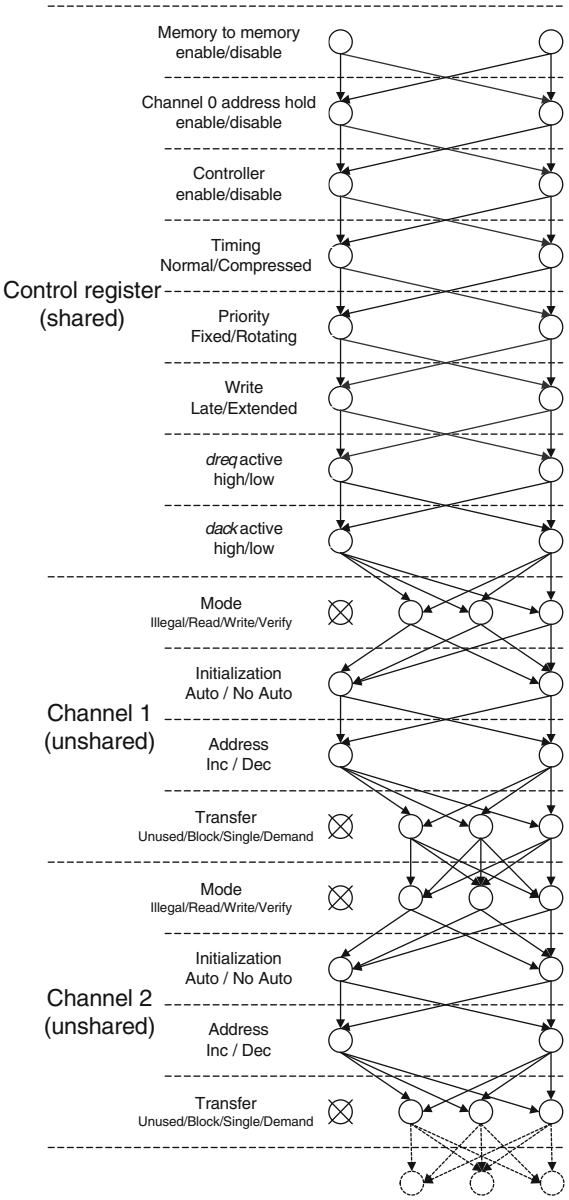

Fig. 5 Part of the configuration graph, representing the modules of an 8237 compliant DMAC. Nodes on the same level represent alternative configurations related to the same functionality. The Control Register is a shared module, while Channels 1 to 4 are unshared. Some of the configurations are prohibited, so the corresponding nodes are not reachable

stimuli being applied by the microprocessor itself, while the complete process is supported by other peripherals interacting with the DMAC.

The system used in the second experiment is depicted in Fig. 6. It contains an 8051 compatible IP core [1], a RAM block, a program ROM, and other peripherals with DMA transfer capability, labeled as P_1-P_4 in the same figure. To connect the processor core to the system bus and provide the DMA module with the proper bus control signals, a

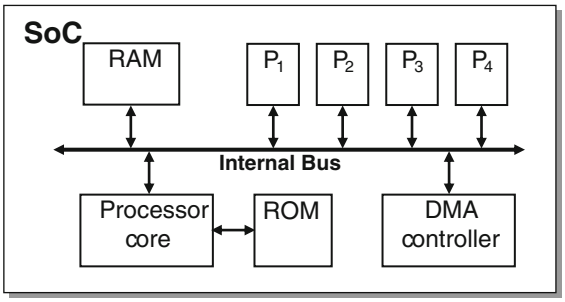

Fig. 6 Microprocessor-based environment used in simulation for verification purposes

simple logic module was interposed between the 8051 external memory port and the data bus. The module includes the required three-state buffers and it filters the addresses generated by the processor core, relying on the fact that the employed processor outputs address 0×00000000 when not requiring the control of the data bus.

5 Experimental Results

Making use of the described methodology a compact verification stimuli set for the DMA controller has been generated. Firstly, the configuration registers, such as the command register and mode register, are identified; secondly, coverage metrics are chosen. Considering the available DMAC model, described in VHDL at RTL, the most important code coverage metrics are statement, condition, and branch coverage. Additionally, a list of corner cases corresponding to operative modes derived from the device specifications is compiled and included in a functional coverage check table. Such table includes, among others, the following modes:

- activation of the four channels in rotating priority mode;
- activation of peripheral-to-memory transfer on at least one channel with introduction of wait states;
- channel 0 and 1 performing a memory-to-memory transfer while enabling the channel 0 address hold.

In the following step, the shared and unshared resources within the controller are labeled and the configuration graph is built from DMAC specifications. 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 the channel control, such as address registers, word count registers and channel mode registers.

By discarding the invalid configurations, we obtained a configuration graph composed of 44 nodes, organized on 24 levels. Considering the shared resources in the DMA descrip-

tion, the data transfer control performed by the DMA priority logic is not directly related to the chosen channel, neither to the address increment or decrement. Thus, each of the transfer modes is linked to only one of the other resources. This is directly mapped on the configuration graph.

Theoretically, by analyzing the device configuration registers, it is possible to identify as many as 2^{32} possible device configurations, which decreases to $2^8 \cdot [2^2(2^2-1)]^4$ considering the illegal modes. On the other side, exploiting the proposed method, 18 configuration paths are obtained..

From each of these paths, a verification procedure is obtained, composed of a configuration part, where the DMAC registers are filled with suitable configuration data, and an operation part, where the device is excited coherently with the specified configuration by acting on its ports.

An example of verification procedure derived from a configuration path is described in the following:

a. The command register is set with values corresponding to: memory-to-memory disabled; channel 0 address hold disabled; controller enabled; normal timing; rotating priority; late write; *dreq* active low; *dack* active high. The register controlling the four channels is set, so that Channel 1 and Channel 4 are configured as: write; auto initialize disabled; address increment; single mode. Channel 2 and Channel 3 are configured as: write; auto initialize disabled; address decrement; demand mode.
b. The memory area where the DMA will write data from the peripheral is cleared.
c. For each channel, the base address register and word counter are initialized. 16 transfers are set for each channel: channels set with address increment will have a base address of 0×0000, while ones with address decrement will have a base address of 0x0010. This combination of values can toggle the first nibble of both the base address register and the word count register. The excitation of the other nibbles of these registers is addressed by other stimuli sequences in the set.
d. A loop is then executed, requesting the transfer on each channel. Control lines are driven by the testbench. In each iteration several channels are activated at the same time, in order to attest the correctness of the rotating priority schedule.
e. Finally, the memory area where data has been written is checked to ascertain the correctness of the performed data transfer.

As mentioned before, the application of the proposed approach left us with a verification set composed of 18 test sequences.

In order to assess the quality of the stimuli set, we consider functional metrics as well as code coverage ones.

Considering functional coverage metrics, a manual analysis of the results confirms that the check table has been completely covered. On the other hand, regarding code coverage metrics, we prepared two different experiments to apply the verification stimuli to the device under verification in simulation: a dedicated testbench, and a verification environment that uses a microprocessor core as depicted in Fig. 6. Clearly, the obtained verification set is properly translated to a set of signals, in the first case, and to a test program suite, in the second case.

The developing process took about one week, considering the graph construction and analysis, and the implementation of the verification procedures.

The developed testbench applies the complete test set in about 3,200 clock cycles, while about 7,000 clock cycles are needed to complete the same operations relying on the microprocessor to apply the stimuli. This is due to the fact that for the application of each single pattern the microprocessor has to execute one or more instructions. In the latter case, the verification program set counts about 913 Bytes on the whole.

All data about coverage metrics have been gathered at the end of the generation process, and the experimental evaluation was performed using Modelsim SE-64 V.6.5c by Mentor Graphics. All the experiments were run on an Intel core2duo processor with 2 Gb RAM.

Interestingly, the same verification coverage figures are obtained independently on the methodology through which the stimuli are applied (dedicated testbench or microprocessor-based system). Table 2 reports the coverage results obtained by the application of the complete verification set: all chosen metrics were completely saturated without the need of iterating the process. Some percentages reported do not reach 100% only because of dead code blocks (for example, unreachable conditional branches).

The results in Table 2 show that the proposed method improves both the outcomes reported in [18], due to a more

Table 2 Coverage results

	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

structured approach, and in [11], thanks to a more efficient graph covering algorithm.

6 Conclusion

This paper presents a functional based approach to the generation of stimuli for design verification of DMA controllers. Following the proposed approach, a verification engineer can develop in a short time an effective and compact verification set able to saturate verification metrics, relying only on a high-level description of the DUT.

The obtained stimuli set can be easily applied in simulation resorting to different strategies (i.e., a dedicated testbench, or a microprocessor based system), depending on the verification environment available for the verification team.

Experimental results, gathered on an 8237-compliant DMA controller, demonstrated that the proposed approach is able to saturate the available coverage metrics, thus assessing the quality of the stimuli set generated.

Future developments include the application of the proposed methodology to other peripherals and using the principles described to enhance VV&T techniques, as well as further automation in the generation of testbench signals and instruction sequences to stimulate the peripheral modules.

References

1. 8051 IP Core circuit description (VHDL): Oregano Systems web site, http://www.oregano.at/eng/
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 Design & Test of Computers vol. 26, n. 2, pp. 52–63
3. Apostolakis A, Psarakis M, Gizopoulos D, Paschalis A (2007) “Functional processor-based testing of communication peripherals in systems-on-Chip,” IEEE transactions on very large scale integration (VLSI) systems, vol. 15, n. 8, pp. 971–975
4. Bergeron J (2003) Writing testbenches: functional verification of HDL models, Second Edition. Kluwer Academic Publishers, Norwell, Massachusetts, USA, p. 512. ISBN: 1-4020-7401-8
5. 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
6. Bose M, Shin J and Rudnick E (2001) “A genetic approach to automatic bias generation for biased random instruction generation”. Evolutionary Congress Proc on pp 442–448
7. Braun M, Rosenstiel W, Schubert KD (2003) “Comparison of Bayesian networks and data mining for coverage directed verification category simulation-based verification,” High-Level Design Validation and Test Workshop, Eighth IEEE International vol., no., pp. 91–95
8. Bushnell ML and Agrawal VD (2000) “Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits”. Springer-Verlag
9. 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
10. Goodenough JB and Gerhart SL (1977) “Toward a theory of testing: data selection criteria, current trends in programming methodology”, vol. 2. In Yeh RT (ed.) Prentice-Hall, Englewood Cliffs, pp. 44–79
11. Grosso M, Perez HWJ, Ravotto D, Sanchez E, Sonza Reorda M, Medina-Velasco J (2010) “Functional test generation for DMA controllers”, 11th Latin American Test Workshop (LATW 2010), pp. 1–6
12. Grosso M, Perez HWJ, Ravotto D, Sanchez E, Sonza Reorda M, Medina-Velasco J (2010) “A software-based self-test methodology for system peripherals”, *15th IEEE European Test Symposium (ETS’10)*, pp. 195–200
13. Piziali A (2004) “Functional verification coverage measurement and analysis”, Springer
14. Pradhan Dhiraj K, Harris Ian G (2009) “Practical design verification”, Cambridge University Press, p. 276
15. Psarakis M, Gizopoulos D, Sanchez E, Sonza Reorda M (2010) “Microprocessor software-based self-testing”, Design & Test of Computers, IEEE, vol. 27, no.3, pp.4–19
16. Ravotto D, Sanchez E, Sonza Reorda M, Squillero G (2009) “Design validation of multithreaded architectures using concurrent threads evolution”, IEEE 22nd Annual Symposium on Integrated Circuits and System Design
17. Rosenberg S (2003) “Combined Coverage Verification Speeds Verification”. EEdesign
18. Sosnowski J, Tupaj L (*2010*) “CPU testability in embedded systems”, Proc of IEEE Int Symp Delta pp. 108–112
19. Strum M, Wang Jiang Chau, Romero EL (2005) “Comparing two testbench methods for hierarchical functional verification of a bluetooth baseband adaptor,” Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS’05. Third IEEE/ACM/IFIP International Conference on, vol., no., pp. 327–332
20. Tasiran S and Keutzer K (2001) “Coverage metrics for functional validation of hardware designs”, IEEE Design & Test of Computers, vol.18: no.4 pp. 36–45
21. Wu Y, Yu L, Xue K, Zhuangr W (1998) “Functional verification of memory controller based on the hierarchical test bench”, Macroelectronics and computer pp. 25–28(2)

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 UniversidadNacional 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.

Danilo Ravotto obtained his PhD at Politecnico di Torino, Torino, Italy. His research interests include test and diagnosis of advanced processors, and evolutionary algorithms. He is a student member of the IEEE.

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.

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.