A Common Platform for Bridging Pre- and Post-Silicon Verification in Mixed-Signal Designs

Dominik Widhalm, Stefan Tauner, Martin Horauer
University of Applied Sciences Technikum Wien
Höchstädtplatz 6, 1200 Vienna, Austria
{widhalm, tauner, horauer}@technikum-wien.at

Achim Schumacher, Alexander Haggenmiller
Infineon Technologies Austria AG
Siemensstraße 2, 9500 Villach, Austria
{achim.schumacher, alex.haggenmiller}@infineon.com

***Abstract*—The steadily increasing complexity of integrated circuits raises the burden on verification engineers of sophisticated System on Chips (SoCs). Simulations and formal verification alone are not able to detect all problems in a design due to unforeseen interactions between modules. These problems may occur even if the modules are logically independent from each other due to physical effects. Bugs found in post-silicon verification in the laboratory on the other hand need to be investigated and fixed with the help of design engineers. Therefore, collaboration between pre- and post-silicon verification is mandatory, which also allows to exploit synergies. Unfortunately, many environments currently in use handle these activities almost in complete separation, which complicates cooperations. This paper introduces a new approach to improve the situation by using a common test description language for pre- and post-silicon verification. It allows to describe test cases to be run in simulation as well as in physical post-silicon environments.**

***Keywords*—Simulation, Testability, Test generation, Verification**

I. Introduction

The permanent advances in manufacturing process technology fuel the steady increase of complexity in modern mixed-signal designs. Shrinking feature-sizes allow to integrate an ever growing number of transistors on the same silicon area. This fact not only causes the well-known *productivity gap* it also makes the proper verification of the design more challenging. Bugs escaping the verification process not only cause a loss of customer revenue but also the reputation of brands may be impaired. In addition to the increasing complexity of modern designs verification teams also have to cope with shortened time-to-market aiming to release the chips as early as possible.

Currently the majority of the verification flows are tailored for either digital or analog designs only. In the domain of mixed-signal verification there is a great need for new innovative test concepts to verify digital as well as analog parts. The analog modules of modern mixed-signal designs cause the main test costs due to non-linearities and parasitic effects on the actual silicon (refer to [1]).

Verification aims at detecting functional as well as physical bugs hidden in the design under verification (DUV). This process can be divided into pre- and post-silicon verification. The former contains a great variety of methods and tools supporting the verification engineers in finding design bugs and deviations from the specification. Post-silicon activities are also capable of detecting physical bugs by debugging and characterizing silicon prototypes. In modern designs the probability of physical bugs is increasing due to the ever shrinking feature sizes and rising complexity of the interactions between analog and digital modules. In contemporary development flows a lot of effort is spent in both areas separately due to the different nature of the respective environments. Furthermore, bugs escape pre-silicon verification which have to be found in post-silicon activities. These bugs often origin in the complex interaction of various components in rare corner cases and in combination with physical aspects.

One approach to ensure functional correctness in an ever decreasing overall verification time is to efficiently use synergies of pre- and post-silicon techniques by the use of proper bridging. Exploiting these synergies helps to reduce verification time and therefore lowers development costs. Using a bidirectional bridging approach further enables a back-annotation of any errors found. Additionally, the proposed approach allows to reuse test cases created for simulation in post-silicon verification, thus preventing redundant work.

This paper presents an approach for bridging pre- and post-silicon verification in the mixed-signal design development process of Infineon Technologies Austria AG. Its main contributions are:

1) Development of a bidirectional bridging approach.
2) Evaluation of the concept using a predefined use case.

The organization of the paper is laid out below. First, the related work is discussed, followed by a presentation of the status quo of the verification of AURIX microcontrollers at Infineon in section III. Next, in section IV, the associated problems and the proposed remedy are set out. The current implementation of this approach is described in section V before the evaluation test case in section VI. At the end, in section VII, a summary of the paper and a short outlook on further work is given.

II. Related Work

Although the gap between pre- and post-silicon verification is a well-known issue, only a few approaches for bridging have been published in the literature so far.

978-1-4799-6144-6/15/$31.00 ©2015 IEEE

In [2] and [3], Adir et al. present a method to create a common platform for pre- and post-silicon verification. The common platform bases on a so-called *Exerciser*. The name *Exerciser* origins in the function of the program since it exercises the DUV by testing predefined scenarios. It runs on the DUV and provides a simulator-like environment for post-silicon verification. As stated in [2], modern *Exercisers* are self-contained solutions since they are able to generate individual test cases for predefined scenarios, run them on the target and do the checking. For this purpose they are equipped with OS-like services which are required by the test cases. Another advantage of *Exercisers* is that they can run in an infinite loop. According to [3] there is a need "to consider pre- and post-silicon efforts as a single effort that is executed on different platforms".

Melani et al. propose an approach in [1] that uses a common description language for creating an environment compatible with pre- and post-silicon verification. This approach enables the reproduction of pre-silicon test cases in a lab environment and features an automatic extraction of the test results which are then compared with the expected values from simulation. To that end, the authors use a common mixed-signal hardware description language (MSHDL), namely VHDL-AMS, to build the simulation environment including not only the system model but also the input signal generation as well as the elaboration of the results. For the lab environment a script extracts all settings required for configuring the DUV, signal generation and data acquisition from the firmware.

The basic idea is to augment the firmware with additional information in form of annotations, which defines how to set up and execute the test in any environment. Therefore, the firmware is used as a kind of glue which links pre- with post-silicon verification. The approach gains a reduction of the overall verification time by automating the information exchange between design and test environments.

Both the "*Exerciser*" and the "*MSHDL*" approach are tailored for a specific area and are not commonly usable. But they offer some valuable considerations and ideas for developing a suitable bridging approach to be used at Infineon Technologies Austria AG.

III. Current Flow(s)

While the proposed approach is generic in nature, it was developed primarily for the verification of Infineon's microcontrollers from the AURIX family. The AURIX is an automotive microcontroller that integrates three of Infineon's TriCore CPUs. Altogether, more than 60 digital, analog and mixed-signal modules are integrated in this SoC. Especially the analog mixed-signal verification of the interaction between analog and digital parts proved to be a very crucial task. Some modules are logically tightly coupled with other modules (e.g., the ADC and the generic timer module (GTM)). Therefore their correct functioning also depends on the proper interaction of these modules.

In order to efficiently achieve a high verification coverage the verification activities have to be appropriately planned in advance. The planning process includes the determination of the verification objectives and also outlines how to achieve them. Up to now this planning is done separately for pre- and post-silicon verification which may hamper the exploitation of possible synergies.

In pre-silicon verification a variety of methods (e.g., simulation and formal verification) is used to verify the design and to check it against its specification. At Infineon primary simulation is used in pre-silicon verification which is applied on different levels of abstraction. On module level the basic functionality of each module is checked, mainly by the use of white-box tests. A subsystem is a group of modules whose functionality is tightly coupled (like the aforementioned ADC and GTM). On subsystem level the modules are treated as black boxes and the focus is on the interfaces and interactions of these modules, assuming that the functionality of the modules have already been properly verified on module level. Chip- and system-level tests consider all modules of the design. Their main intent is to test realistic use cases (which are sometimes even provided by the customers).

Simulation requires the presence of a suitable testbench. Traditionally, testbenches are mainly written by hand resulting in a relatively high creation time. To lower this effort and to support the simulation process a powerful SoC-level verification environment was developed at Infineon which is named *AGENtiX*. It provides front-end tools, C header files, a basic library for functions and other testbench elements for the mixed-language simulation of Infineon's automotive microcontrollers. The testbenches as well as the test cases are written in SystemC allowing to use abstract models of modules and bus functional models (BFMs) for the purpose of hardware/software co-verification. *AGENtiX* is further able to simulate multiple CPUs working in parallel. Additionally, it offers a virtual communication network to control the single modules of the testbench. In *AGENtiX*, analog and mixed-signal modules are included as abstract BFMs also connected to the communication network.

Post-silicon verification additionally focuses on issues not covered by pre-silicon tests (e.g., issues on the real silicon introduced by process technology variations). The tests are executed on real-silicon prototypes, which are most often assembled on an engineering board. Stimuli applied to the DUV are either generated by power supplies (for simple signals) or by pattern generators. In contrast to pre-silicon simulation where all necessary parts are included in the testbench, in post-silicon verification additional lab equipment is needed. To drive these control and measurement instruments and to automate parts of the verification process, a tool named *JAZZ* was developed at Infineon. *JAZZ* follows a layered approach for data and control abstraction and supports a wide range of standardized communication interfaces (e.g., JTAG, GPIB, or Infineon's Device Access Server (DAS)) to control the lab equipment as well as for the communication with the DUV.

Pre- and post-silicon test cases are divided in three main categories depending on their stimuli creation concept.

1) The test concept traditionally used in simulation are

directed tests. In directed tests (sometimes also referred to as focus tests) the exact stimuli values are predefined and mostly written by hand. Directed test cases have a deterministic sequence of transactions (such as stimuli creation). Therefore, repeating directed tests will always obtain the same result. These tests cover scenarios, which are often related to user applications or pre-known corner cases.

2) In random tests the stimuli are generated by transaction generators in form of pseudo-random sequences of transactions and data. This random generation is only limited by predefined rules that ensure that only legal stimuli are applied to the DUV. The ordering of transactions as well as their contents changes with different seeds initializing the (pseudo) random generator. Therefore, these test cases can be repeated several times (with different random starting seeds) each with different resulting stimuli. Due to these semi-automatic generation of stimuli more input scenarios can be covered than in directed tests. To check the response of the DUV reference models are used.
3) Directed-random tests are a hybrid of directed and random tests that also employ random stimuli generation. The stimuli generation for directed-random tests is limited by a set of constrains (such as range limits or allowed transaction schemes). Therefore these tests are also called constrained-random tests. This allows to automatically create stimuli that contain random variations within user-defined limits (e.g., randomized parameter values for a specific opcode of a processor). Repeating directed-random tests with different seeds will result in the same mixture of transactions, but with different ordering and parameters.

IV. Proposed Methodology

As can be seen in the previous section, Infineon uses a number of methods to verify their products in appropriate ways. During the status quo analysis it was revealed that there are two areas offering great potential for improvements:

1) The flow of information between the individual methods.
2) The simplification of test entry in post-silicon verification.

There is a lack of communication and bridging efforts between the individual pre- and post-silicon verification steps. Furthermore, the automation in post-silicon test case creation needs to be improved. Currently, a lot of effort is spent recreating basic test cases for each DUV although the tests are similar for all new products or models. This effort can be minimized by the use of already existing test cases (e.g., from simulation) and improving code reuse. If this idea is further considered, the creation of a common platform for the definition of shareable test cases seems promising. Test cases would have to be defined and written only once, resulting in a significant reduction of overall verification time. Furthermore, if an error in the test sequence is found during its execution, or if the test case is modified, only the common test case description needs to be changed. These changes can then be distributed automatically by regenerating the environment-specific test cases.

The basic concept of the proposed approach is shown in figure 1. Essential elements (printed in *italics* below) are discussed in the remainder of this section, except for the actual *converter* and *interpreter modules* that are presented in section V.

A common environment as well as a common language are used to describe the test cases and to share them between the single activities. The most important part of the proposed approach are the *user input* files, here especially the *human-readable test case*. This common format allows to define test cases once that are further usable in pre- and post-silicon activities. Additionally, legacy environments can be supported by exporting test cases appropriately.

The common test description has to be written in a general, platform-independent form to be universally usable in all environments. However, since it is necessary to define a mapping between the signals described in the test cases and their execution environment, there are also *configuration information* files, which are partially environment-specific. They might also include information that is relevant for some of the environments only, e.g., fixed temperature profiles used for testing the real silicon that are not used in simulation. Before and while test execution this *configuration information* is used together with *environment-specific information* (e.g., register addresses, code templates or instrument definitions) to describe the test environment.

Since test cases often require complex sequences, the use of a language that supports constructs like loops or branches is required. Several options were examined and it was determined to use a scripting language (e.g., Perl or Python) to describe test cases. This allows to define tests at a high level of abstraction and additionally provides the option to store and process data. Information can not only be stored in simple variables, but also in complex data structures (e.g., hash maps) allowing to store large amounts of data in a structured way.

The use of a scripting language has another advantage, namely the ability to easily randomize stimuli creation. It is therefore possible to create directed, random, and directed random test cases to achieve a maximum test coverage (refer to section III).

To spare test engineers from rewriting often used test sequences manually each time, such sequences are stored in *test case element* pools. These libraries also contains standard functions (e.g., calculations) and universally applicable information (e.g., description of complex waveforms).

As revealed by an initial prototype, an important role in our concept plays the synchronization of the test software and the DUV. If an accurate synchronization is not minded in the test case creation, it can significantly slow down the test execution or end in wrong results. Therefore, convenience functions are provided that create barriers that synchronize the execution of the test sequence.

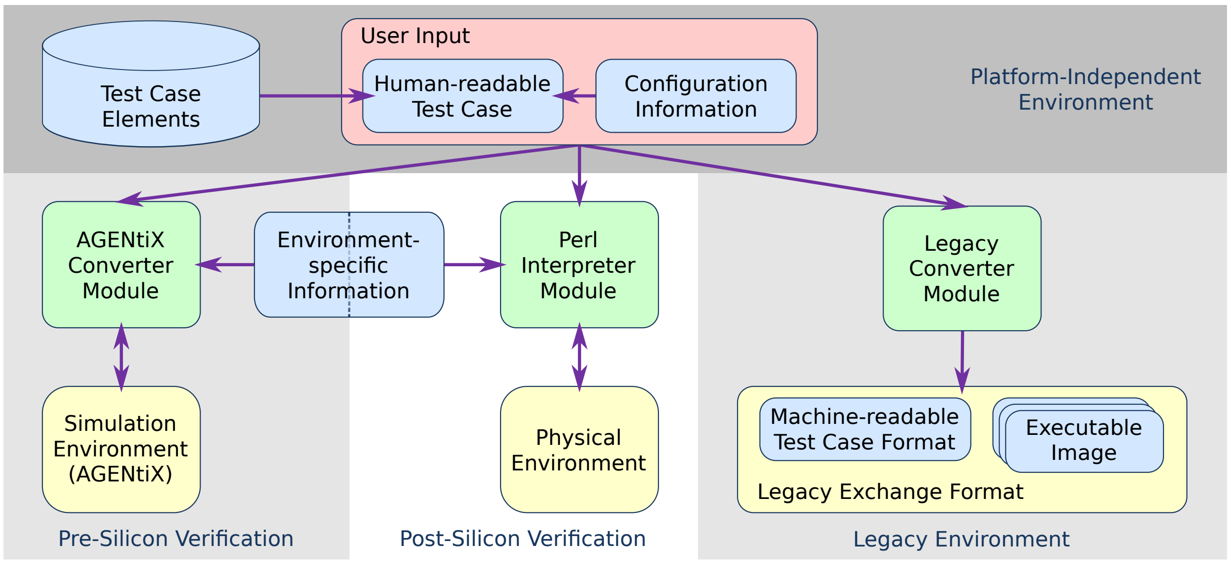

Figure 1. Basic Concept of the Proposed Approach

Another issue related to synchronization is introduced by the parallel execution on multiple targets and in some environments where no dedicated communication channel between the controlling host and the DUVs is available or where it is too slow for real-time interactions. If the test flow depends on intermediate results received from the DUVs the test software must somehow decide when and how to proceed at such arbitration points. The preliminary idea to solve this is to distinguish between two basic types of platforms, referred to as primary and secondary platforms. Primary platforms allow to execute the test case and communicate with the host in real-time, while secondary platforms do not and therefore require another solution. To that end, the test cases need to be written with these circumstances in mind. They must not contain any communication instructions and use sufficient wait times to accommodate for variations in execution times. Test results are stored using on-board resources and transfered to the test software after the tests have finished.

V. Implementation

As explained in the previous section, describing test cases with the help of a scripting language is advantageous. The implementation described hereinafter uses Perl 5 because it is already in heavy use at Infineon and therefore well supported in the target working environment.

The test engineer can rely almost exclusively on the provided testbench elements and helper classes, but nevertheless some knowledge of Perl is required. The test cases are loaded by the implementation in a way so that all basic testbench elements are available and do not need to be imported. Additionally, it is possible to include foreign Perl packages (e.g., from CPAN) if need be.

To actually design a test case the engineer needs to create instances of the respective DUV and (abstract) laboratory instruments which are then connected appropriately. In physical environments this needs to be done manually by the laboratory operators according to a connection scheme stored in additional mapping files that link the physical device addresses with their virtual representation in the test cases. In simulations these pin mappings are stored together with the test cases and the information is automatically extracted.

Within test descriptions stimuli can be applied by directly interacting with the instrument objects and calling setter functions of the respective outputs and using Perl's control flow statements to control the execution. Alternatively there exists a sophisticated scheduling framework that works with shapes and events. In this framework the test engineer can easily create complex shapes by combining basic signals like sinusoids, sawtooths etc. Additionally, one can define that actions are triggered on certain points of these shapes or synchronized to points in time. When executed in the post-silicon environment there are severe limits to this complex approach, for example it is impossible to coordinate a sinus output of a function generator with a directly controlled power supply. In these circumstances the test engineer needs to evaluate if imperfections in synchronization are tolerable or if the course of events can be described appropriately.

The actual interpreter and converter modules are specific to the environment. For this reason they are described separately in the following subsections.

A. Test Setup for the Physical Environment

In the post-silicon environment the implementation relies on Infineon's *perlcore* for communication with the DUV and laboratory instruments. *Perlcore* internally uses another Infineon tool named DAS to access the DUV. DAS acts as a proxy between hardware (e.g., a debug interface) and software tools and is available as DLL for Microsoft Windows. Besides wrapping DAS functions *perlcore* also provides an interface to laboratory instruments with the help of VISA (an industry standard API).

To test the prototype implementation an Infineon TriBoard with an AURIX TC277TE was used. It features an on-board debugger interface accessible via USB and allows to attach four break-out boards that expose almost all pins of the microcontroller. This makes an easy setup possible where only power and one USB cable need to be connected in addition to test case-specific peripherals.

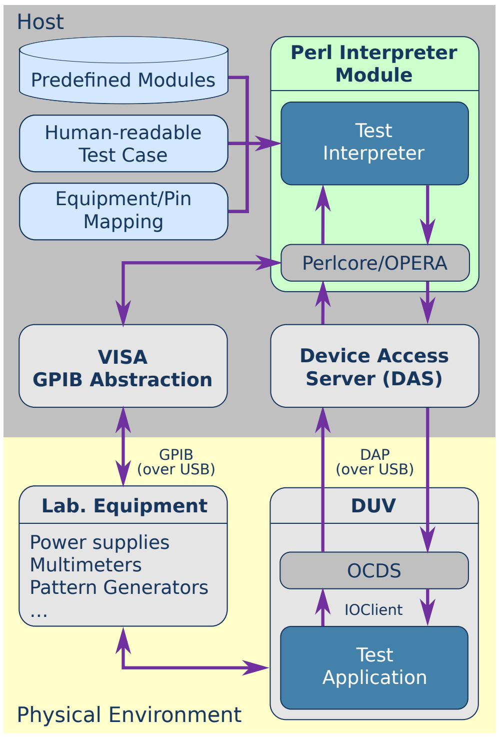

Figure 2. Information & Control Flow in Post-Silicon Verification

Figure 2 depicts the information flow in the post-silicon environment, showing the already explained input files (top left). Our test case interpreter manages the execution of the tests by sending commands via *perlcore* to laboratory equipment and the firmware on the DUV. Communication with the debugger module of the DUV (i.e., the On-Chip Debug Support (OCDS)) is established over the Device Access Port (DAP) of the microcontroller which is a two-pin debugging interface developed by Infineon. The DAP protocol is transmitted via a USB-to-serial converter mounted on the board.

Interaction with the test firmware uses a feature of the OCDS named *IOClient* that allows to communicate asynchronously with the firmware running on the microcontroller without disturbing its progress by a simple mailbox-like mechanism. In one of *IOClient*'s modes (i.e., *Communication mode*) any read or write from the host results merely in a bit flip in a flag register that can be polled by the firmware or, if requested by the application, an interrupt is triggered. The firmware is then free to decide when to fulfill the request. Additionally, the *IOClient* mode of the OCDS provides a deterministic way to reset the communication between the host and the firmware. This is used for the initial synchronization of the DUV with the host at the beginning of a test run.

The current firmware is tailored for each test case and synchronizes with the host software by using *IOClient* transfers. Any measurements or other results are sent back to the host via IOClient as well.

B. Code Generation for the Pre-Silicon Environment

For simulations the test cases need to be converted to SystemC files to be fed into the AGENtiX environment. To that end, a global configuration variable is set at startup depending on the target environment to influence the runtime behavior of the Perl scripts. If the variable indicates a simulation environment then the low-level functions no longer control laboratory instruments or the DUV but instead emit SystemC code that interacts with the various BFMs representing the DUV and testbench elements. The resulting code can then be simply compiled and simulated by AGENtiX.

Additionally to the simulation and post-silicon environments our approach allows to support legacy test flows as well by converting the test cases to previously used formats. This works similarly to the code generation for AGENtiX as explained above.

VI. Evaluation

A direct comparison between the previous workflow and the one implied by our approach is hard for multiple reasons. First of all, a suitable metric is not obvious: the most interesting one would probably be to measure the speedup of development and bug hunting relative to the previous work flow. This however greatly varies with the encountered design flaws and is hard to reproduce. It would also require a complex test setup where engineers from all affected departments within Infineon try to find the same bugs with different approaches. Furthermore, to model any bugs realistically in this scenario one would have to produce faulty silicon on purpose. Implementing such an evaluation scheme would have exceeded our resources. Alternative metrics like performance or size of the generated code would be more easy to compare but do not reflect the intrinsic advantages of our approach.

We have therefore focused on verifying the feasibility of the approach itself. We have chosen some existing test cases to be reimplemented with our approach. Instead of separately handcrafting respective programs for the simulation and post-silicon environment the code is only written once.

One of these test cases characterizes some aspects of the ADC unit found in AURIX microcontrollers as explained below. It is based on the well-known and widespread ADC test concept known as *"histogram method"* (refer to [4] and [5]) and will be further referred to as *histogram test*. The flow diagram of the histogram test is depicted in figure 3.

By the use of the histogram test the standard deviation of an ADC can be evaluated which gives a good overview on the stability of its conversion results. The test proceeds as follows:

1) Apply a constant voltage to desired input channel.
2) Trigger a few thousand conversions.
3) Build a histogram from conversion results.
4) Derive the standard deviation from the histogram.

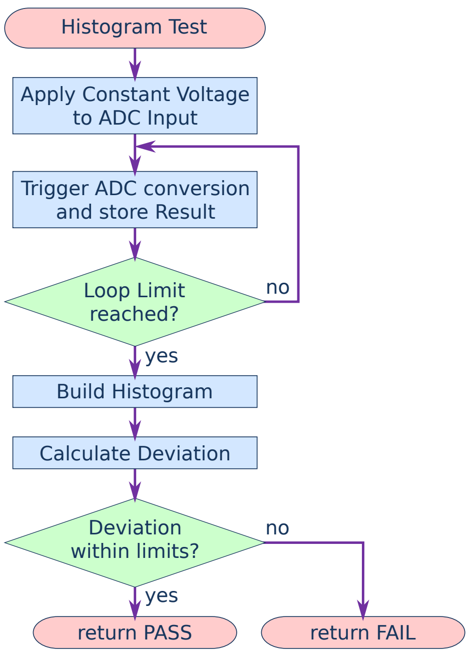

Figure 3. Flow Diagram of the Histogram Test

To build the histogram the occurrence of every possible code word is counted. In an ideal ADC only the one code word corresponding to the applied voltage level would occur. Because of noise also adjacent code words appear in real ADCs. The counts of the code words are normally Gaussian-distributed where the variance, and respectively the standard deviation, depends on the quality of the actual ADC.

This represents a typical test case that is done in the laboratory after manufacturing to evaluate a mixed-signal design. The circumstances leading to unexpected results (e.g., a conspicuously high variance of measurements) can then (hopefully) be reproduced and investigated in simulation to find design flaws. Our approach tremendously simplifies the step needed to generate the code for these simulations.

VII. Conclusion and Outlook

Due to the increasing gap between complexity of designs and suitable ways to verify them, a significant optimization of the verification process will be inevitable. By using a common description language, test cases can be defined once for all environments, and thus help to noticeably reduce the overall verification time. For obvious reasons, this requires a change in the verification planning process, namely the creation of a unified process in which engineers of all relevant areas take part.

The approach presented in this paper allows to share test cases between different pre- and post-silicon verification activities. This is achieved by creating the test cases using a common description language. The common test descriptions can then be automatically converted to environment-specific test cases. This offers the advantage that test cases need only to be written once and can then be commonly used in different environments. Another benefit is that any changes have to be done in the common format only.

The feasibility of the proposed approach was verified using predefined test cases. It shows that test cases can easily be defined using the capabilities of the proposed description language.

Currently the test firmwares running on the DUVs are static only, i.e., they need to be tailored to each test case and are merely downloaded to the target. Improving this limitation will be the main focus of future work. One solution to this could be to implement a virtual machine running on the DUV that interprets abstract representations of a test flow. The main advantage of this approach is that the firmware only needs to be flashed once. Alternatively, the firmware could be generated from the test description on each run. In any case, there needs to be a way to express the desired instructions of the DUV within the test description language.

The code generation for AGENtiX by simply changing the semantics of the low-level functions used in test cases produces very verbose source code. For example, where test descriptions define a simple loop the generated code contains the unrolled loop without explicit counter variables and abort conditions. A more advanced implementation could transform the so-called *op tree* generated by the Perl compiler into a more usable Abstract Syntax Tree (AST) and base the SystemC code generation on that. This would also make the implementation of back annotations easier, e.g., to allow propagation of build errors back to the test description code. Moreover, it would remove serious limitations in the expressibility of test descriptions, e.g., it would enable branches that need to be evaluated at runtime.

Acknowledgment

This work has been supported by Infineon Technologies Austria AG and by the publicly funded R&D project Josef Ressel Center for Verification of Embedded Computing Systems (VECS) managed by the Christian Doppler Gesellschaft (CDG).

References

[1] M. Melani, F. D'Ascoli, C. Marino, L. Fanucci, A. Giambastiani, A. Rocchi, M. De Marinis, and A. Monterastelli, "An Integrated Flow from pre-Silicon Simulation to post-Silicon Verification," in *Ph. D. Research in Microelectronics and Electronics*, 2006, pp. 205–208. http://dx.doi.org/10.1109/RME.2006.1689932

[2] A. Adir, S. Copty, S. Landa, A. Nahir, G. Shurek, A. Ziv, C. Meissner, and J. Schumann, "A unified methodology for pre-silicon verification and post-silicon validation," in *Design, Automation Test in Europe (DATE) Exhibition*, 2011, pp. 1–6. http://dx.doi.org/10.1109/DATE.2011.5763252

[3] A. Adir, A. Nahir, G. Shurek, A. Ziv, C. Meissner, and J. Schumann, "Leveraging pre-silicon verification resources for the post-silicon validation of the IBM POWER7 processor," in *48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2011, pp. 569–574. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5981978

[4] F. Azaïs, S. Bernard, Y. Bertrand, and M. Renovell, "A Low-Cost BIST Architecture for Linear Histogram Testing of ADCs," *Journal of Electronic Testing*, vol. 17, no. 2, pp. 139–147, April 2001. http://dx.doi.org/10.1023/A:1011173710479

[5] F. Azaïs, S. Bernard, Y. Bertrand, and M. Renovell, "Implementation of a linear histogram BIST for ADCs," in *Design, Automation and Test in Europe (DATE) Proceedings*. IEEE Press, 2001, pp. 590–595. http://dl.acm.org/citation.cfm?id=367072.367829