Test Your Test Programs Pre-Silicon:
A Virtual Test Methodology for Industrial Design Flows

Sebastian Pointner* Oliver Frank† Christoph Hazott† Robert Wille*
*Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
†Infineon Technologies Austria, Linz, Austria
Email: {sebastian.pointner, robert.wille}@jku.at {oliver.frank, christoph.hazott}@infineon.com

***Abstract*—The ever increasing complexity of modern circuits and systems remains a big challenge for the semiconductor industry. Since the life cycle of new products is getting smaller and smaller, companies have to speed-up their design flows to stay competitive. This particularly holds for the efforts spent to verify and test a chip. Here, the development of proper test programs to be executed on the fabricated chip constitutes a serious bottleneck. This is because the first application of the test program on actual silicon frequently unveils errors that need to be addressed – causing debugging loops and a threat to time-to-market objectives. Consequently, it is tried to conduct these tests earlier in the design flow, i.e. before first silicon is available. In this work, we propose a corresponding *virtual test* methodology which allows to test a test program on a virtual representation of the chip (e.g. a SystemC description which is available early in the design process anyway). In contrast to previously proposed solutions, our methodology can be integrated in a generic and black-box fashion into existing flows, i.e. the user does not need to know whether the test program is executed on actual silicon or its virtual description. A case study within an industrial environment confirms the benefits of the proposed methodology.**

I. Introduction

The design and realization of electronic systems belongs to one of the most complex tasks in Electronic Design Automation. To cope with the ever increasing complexity, an elaborated design flow emerged in the past which heavily relies on abstractions and is sketched on the left-hand side of Fig. 1: Starting with the specification, first an initial algorithmic implementation of the desired system is implemented at the *Electronic System Level* (ESL) [1] (using languages such as SystemC [2]). This allows to conduct simulations and, by this, provides the basis e.g. for software/hardware partitioning. After the respectively resulting software and hardware components have been extracted, the implementation of the respectively needed *Application Specific Integrated Circuits* (ASICs) starts – first conducted at the *Register Transfer Level* (RTL; using languages such as VHDL [3], Verilog [4], etc.) and, afterwards, synthesized to a gate level netlist. The resulting descriptions are eventually prepared for fabrication (e.g. involving tasks such as place, route, technology mapping, etc.) and send to a wafer fabrication which realizes the desired chip (the so called tape-out).

Besides the actual design and fabrication process, checking the correct and intended behavior of the desired system is getting even more important. In fact, the efforts to be invested in order to design and realize a system more and more move from design to test and verification, i.e. more time is spent on checking whether the developed ASIC is correct or not. Accordingly, methods for validation and verification are applied through all abstraction levels in the design phase (see e.g. [1], [5], [6]), while, after fabrication, methods for *Automatic Test Pattern Generation* (ATPG, see e.g. [7], [8]) and *test* are common (see e.g. [9], [10]). In this work, we focus on the development of corresponding tests after the chip has been fabricated (i.e. post-silicon test).

Here, the fabricated chip (in the following denoted *Device Under Test*; DUT) is employed within an *Automatic Test Environment* (ATE) which allows to apply dedicated test stimuli to the chip. The stimuli are thereby generated e.g. directly from the user, but the ATE also allows to apply corresponding stimuli obtained e.g. from other components, sensors, etc. Vice versa, the ATE provides interfaces which map the output signals produced by the DUT to the corresponding user outputs, actuators, etc. Moreover, the user's view is limited to the ATE for testing the DUT, as the ATE represents the entire system. Overall, this allows to test the DUT using different scenarios and considering internal as well as external inputs such as sensor data, prepared data to emulate specific use cases (e.g. corner cases), or random test data to observe the systems behavior. To automate these tests, usually a *test program* is applied. A test program represents a collection of different test methods – each of which used to test a different part, aspect, or scenario of the DUT.

However, the development of proper test programs is non-trivial. To some extent, a good test program requires a re-consideration of all major aspects considered during the actual design flow in order to make sure that all properties and characteristics of the DUT are properly tested. Moreover, the development of such a test program is special due to the fact that it can usually not be checked before *first silicon* is available, i.e. before the ASIC has been fabricated. This is sketched on the bottom-left of Fig. 1. Although the test program development can already start in parallel to the design process (i.e. pre-silicon), it cannot be completed before it has been checked on the actual silicon. Moreover, the first application of the test program on actual silicon frequently unveils further problems that need to be addressed and usually require substantial changes in the test program. This causes a bottleneck in the design flow and frequently leads to serious threats with respect to the satisfaction of time-to-market objectives for industrial designs.

In this work, we propose a virtual test development methodology for industrial designs which addresses this problem. The main idea is as follows: Instead of waiting for first silicon in order to check the test program, we use a virtual description of the DUT. Therefore, we are exploiting the fact that several descriptions of the DUT are available as a result of the design flow anyway. More precisely, as sketched on the right-hand side of Fig. 1, we are using the ESL description of

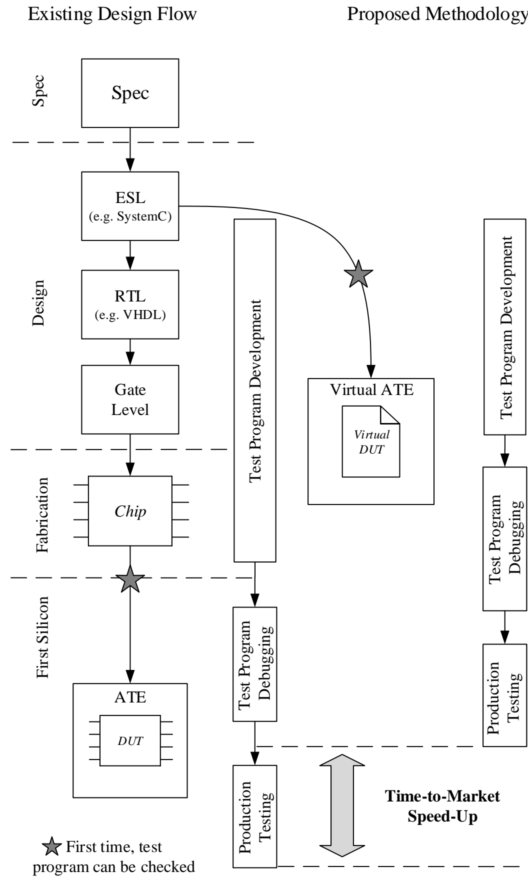

Fig. 1: Today's design flow (w/ proposed virtual test flow).

the design as a virtual representative of the DUT which is not available yet. This representation is integrated into the ATE and subjected to the test program. This allows to conduct tests of the test program prior to first silicon and, by this, overcomes the bottleneck in today's industrial flows. Moreover, compared to related work (covered in detail later in Section II-B), the proposed methodology satisfies industrial requirements of a generic solution which can easily be integrated into existing flows and works in a "black box"-fashion (i.e. the user does not need to know whether the test program is executed on actual silicon or its virtual description).

Case studies within an industrial environment demonstrate the benefits of the proposed methodology. Although executing a test program on a virtual representative consumes more run-time than on an actual DUT in silicon, the availability of a virtual test environment allows to conduct these runs prior to silicon and, by this, weeks or even months before fabrication. Furthermore, due to the "black box"-integration, the user can conduct these checks in the same fashion as he/she would do it with an actual silicon implementation. As a result, we are able to pass through the bottleneck discussed above – a major step towards a more reliable satisfaction of time-to-market objectives.

The remainder of this work is structured as follows: The next section briefly reviews the development of test programs and discusses related work. Afterwards, the main ideas proposed in this paper are outlined and details on their implementation are provided in Section III. Section IV demonstrates the application of the proposed virtual test development flow by means of an industrial case study. Finally, the paper is concluded in Section V.

II. Background

In order to keep this work self-contained, this section briefly reviews the development of test programs and the resulting bottleneck. Afterwards, we discuss the works which have been proposed thus far to address this problem.

A. Test Development

For the automation of the post-silicon test process, modern ASICs are tested by applying so called *test programs*. The development of such *test programs* is dependent on the applied technology in which the considered ASIC shall be realized as well as on the requirements given by the respectively used ATE supplier. *Test programs*, as they are widely used within industry, are realized using frameworks based on programming languages such as *Visual Basic* or *C++*. Such frameworks extend the basic functionality of those languages with advanced support which is needed for industrial semiconductor testing. Examples for extensions can be found in class libraries (e.g. for the evaluation of tests) or additional tools (e.g. visualizations of instruments). Based on those extensions, a comprehensive framework for the entire test program development cycle, as well as for the test process in the wafer fabrication is supported by the ATE supplier.

The development of the test program can be conducted in parallel to the actual design of the considered ASIC. The goal is the realization of a program which (1) automatically applies several tests on the eventually realized ASIC (after a first silicon realization of it is available) and (2) checks whether the intended outputs are obtained. Typical tests contain e.g. checking the functionality of signal processing units, the correct behavior of digital interfaces, or the internal memory embedded into the ASIC.

However, whether or not a test program properly conducts the intended tests is hard to check before the first silicon is available. In fact, only "sanity checks" such as whether the correct syntax has been used can be conducted beforehand e.g. by the correspondingly used compilers and/or interpreters. For all other checks, a silicon implementation of the ASIC needs to be available. This is crucial considering that test programs are frequently implemented with unexpected behavior. Such unexpected behavior can have a wide spectrum of reasons. Those reasons can yield from wrong configurations of test instruments or from assumptions made during the test program (since e.g. an erroneous instrument configuration cannot be detected without applying the instrument). Examples of typical errors not detected prior silicon include:

- Wrong DUT configuration (e.g. wrong register configuration)
- Wrong test implementation or test specification (e.g. wrong test-instrument configuration)
- Discrepancy between the test specification and the device specification
- Behavior of ASIC differs from the expected behavior described in the datasheet

Once such errors have been detected, they of course need to be debugged and fixed. Because of the ever-increasing complexity of ASICs and, hence, also corresponding test programs, this usually requires a significant amount of time (frequently delaying the product release not only by some days, but even weeks in many cases). This bears a significant risk to time-to-market objectives and is particularly unfortunate, since plenty of time to implement a correct test program is usually available pre-silicon, i.e. when the actual ASIC is in its design phase.

B. Related Work

The problem discussed above constitutes a serious bottleneck that affects all major players in the semiconductor industry, as e.g. shown by corresponding works e.g. from *Analog Devices* [11], *Teradyne* [12], and *Infineon Technologies* [13], [14]. But, thus far, no satisfactory works exist which tackles this problem in an industrial-strengthened fashion. Of course, an obvious idea is to lift checking and debugging the test program to an earlier, i.e. pre-silicon, stage – motivating a so called *virtual test*.

To the best of our knowledge, the first approach in this direction has been presented by Teradyne in [12]. Here, it has been suggested to unify the verification of the design process of the ASIC as well as the of the test program development. Later, works such as [13] or [14] have been proposed, which refined this idea. In all these cases, a pre-silicon representation of the DUT has been considered in order to check the test program (e.g. a prototypical FPGA implementation in case of [14] or an RTL description in case of [13]). Both directions, however, come with serious shortcomings. More precisely:

- Following approaches such as proposed in [14] requires a DUT replacement e.g. in terms of an FPGA implementation. Although easier to create than an ASIC in silicon, this nevertheless requires an actual realization which still is available rather late in the design flow (usually, after an RTL description has been completed). Moreover, the package of the resulting FPGA often will not match with the package of the DUT – causing complications with the respectively applied ATE that have to be sorted out manually. Finally, FPGAs are designed for digital designs only, making the handling of analogue aspects of the DUT tedious[1].
- Following approaches such as proposed in [13] does not need explicit hardware, but relies on a given RTL model which can be simulated using tools such as ModelSim [15]. However, this significantly differs from the common application scenario a test program developer is used to. In fact, instead of executing his/her program within an ATE, a dedicated tool chain needs to be created that allows for the simulation of the RTL description while, at the same time, inputs/outputs of the (in many cases non-trivial) test programs are instrumented on it. As a result, the user has to deal with several software tools, intermediate formats, and a rather static communication between the DUT (represented in RTL) and the test program, while, at the same time, he/she is actually supposed to check the test program.

[1] In [14], the latter problem has been addressed by outsourcing the analogue parts of an ASIC to a PCB; again requiring substantial manual adjustments.

III. Proposed Methodology

In this work, we propose a virtual test methodology which addresses the shortcomings discussed above. Following the virtual test approach, also our methodology replaces the actual DUT (provided in silicon) by a *virtual DUT* which is available earlier in the design flow. In our case, we use a SystemC implementation as this provides an executable description of the desired functionality which additionally is available early in the design process (i.e. the test program can be checked with it rather early)[2]. But in contrast to the approaches discussed above, we afterwards do not rely on a prototype (to be realized first) or a dedicated simulation tool-chain (in which the virtual DUT additionally has to be instrumented). Instead, we explicitly revisited all steps that are usually conducted within an ATE when executing a test program on actual silicon and developed corresponding *virtual*, i.e. software, solutions for each of these steps. This allows for the realization of a generic and black-box virtual test environment which can easily be integrated into existing flows and for which the user does not need to know whether the test program is executed on actual silicon or its virtual description.

In this section, the details of the ATE realizing this virtual test methodology are provided. First, the steps which are usually conducted to execute a test program are reviewed. Based on that, the corresponding virtual test realization is described afterwards.

A. Execution of a Test Program

Recall that the main task of conducting a test is to execute a test program (provided in software) on the actual DUT (provided in silicon, i.e. hardware). In order to correlate the software with the hardware, an ATE is employed which is structured as shown in Fig. 2. Here, the test program is embedded into a corresponding *Test Software* while the DUT is embedded into a corresponding *Test Hardware*. The interface between those is realized by an *ATE Interface* (provided in software) which, in turn, is connected by a *bus* to the hardware. More precisely, for the execution of any operation on the DUT, the test program utilizes this interface which represents the access point for all DUT operations.

The ATE interface acts thereby as an abstract representation for all instruments offered by the ATE tester (e.g. powering the DUT using a *Power Supply*, capturing an analog signal using an *Analog Capture* unit, applying an analog stimulus using an *Analog Source* unit, or applying a digital stimulus by applying the *Digital Source* as indicated by the respective boxes in the ATE interface depicted in Fig. 2). In this sense, the tester software can be seen as a kind of IDE for the test program which can execute the implemented program. Despite of its execution capabilities, also additional features such as advanced means for the evaluation of the failed tests (statistics) are part of the tester software.

Next, the test program's instructions have to be transported to the DUT (through the tester hardware). To this end, a *bus* connection as already mentioned before is used. This *bus* connection is based on industrially standardized bus protocols (e.g. *General Purpose Interface Bus*; GPIB [16]) and realized stationary.

[2] Note that the concepts of the proposed methodology can similarly be applied e.g. to an RTL description as well.

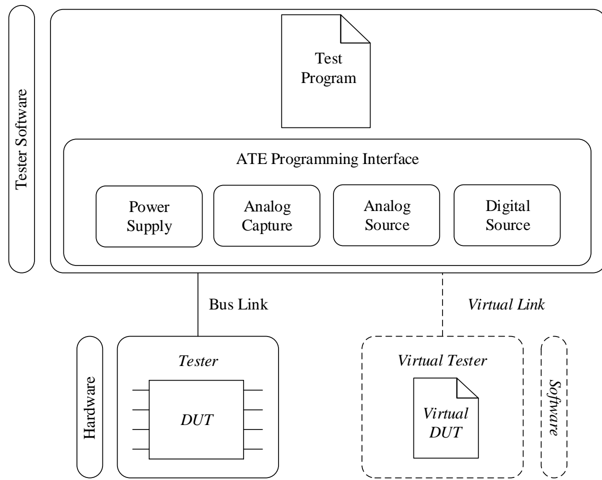

Fig. 2: Structure of the ATE (w/ proposed virt. realization).

A stationary *bus* connection in this context means that both the tester hardware and the workstation which executes the tester software cannot be replaced easily. They are connected e.g. by a physical wire connection which cannot be replaced on the fly.

Finally, the test program's instructions are executed on the DUT by the tester hardware. To this end, the tester hardware is composed of multiple components realizing the test instruments mentioned above (i.e. the power supply, the analog capture, the analog source, or the digital source) in hardware. By this, all test instruments are connected to the DUT and can be configured and applied according to the test program. After the execution, the respectively obtained outputs are transported back through the bus and its interface to the test software, which evaluates the results and conducts the next steps based on them. Overall, this provides an ATE structure allowing the execution of complex tests on the DUT.

B. Realization of a Virtual ATE

The complexity of today's test programs frequently requires intensive checks and debugging steps which, following the structure reviewed above, cannot be conducted before first silicon is available. In the methodology proposed in this work (as introduced above), this shortcoming is addressed by using a virtual DUT (provided e.g. in SystemC) which, nevertheless, shall be instrumented and tested in exactly the same fashion as before (i.e. the user does not know whether he/she tests an actual DUT or a virtual one). Since the hardware tester and all its components cannot deal with a virtual DUT, this requires the realization of an entire software-based *virtual tester* which is sketched in the bottom-right of Fig. 2 and described in this section.

In general, the virtual tester has been realized by implementing all functionality provided by the bus as well as the hardware tester in terms of software. This required significant adjustments to the respective layers of the ATE, namely:

- As already mentioned, the DUT itself is replaced by a virtual DUT.
- Since the virtual tester cannot be connected to the workstation executing the tester software by means of a firm *bus* connection, a corresponding *virtual link* is needed.
- Since the interface suddenly has to support two targets, an actual hardware tester and a virtual tester, the ATE interface needs to be adjusted to allow for both settings (leading to a dedicated *test target selector* as well as several changes to the tester software).

In the following, these issues are discussed in more detail with respect to each layer.

Virtual DUT: The virtual DUT is designed as an ESL model which is capable to replace the actual DUT. Since this task is part of the design flow anyway (cf. again Fig. 1), we can obtain the virtual DUT basically "for free". Compared to the silicon-based DUT, the virtual DUT is implemented in software only. The only extensions which have to be added are related to the communication between the test program and the *virtual DUT*. This is covered by the next layer.

Virtual Tester: The virtual tester is responsible to make the virtual DUT visible for any communication. More precisely, it builds a wrapper around the virtual DUT to allow for the communication between the test program and the virtual DUT. To this end, a communication protocol is employed which has been broken down to three basic operations:

1) Write Request:
   Applies a certain value to a certain pin.
2) Read Request:
   Reads the value of a certain pin and sends the answer back to the test program.
3) Run Request:
   Executes the virtual DUT for the requested time (using the SystemC function call *sc_start(time, unit)*).

Virtual Link: The virtual link is used to connect the test program to the virtual tester. In contrast to the firm bus link from the hardware solution, this virtual link can be realized in a rather flexible fashion. To this end, a realization based on TCP/IP sockets is employed. This realization allows to establish a socket communication between the tester software (realized as desktop application), the virtual tester (realized in C++), and the virtual DUT (realized in SystemC, a C++ library) which, due to available operating system libraries (i.e. BSD or Win32 sockets), all provide proper "off-the-shelf" TCP/IP solutions. Moreover, the flexibility offered by TCP/IP even allows to completely abstract from the location of the workstation executing the virtual tester and, by this, the virtual DUT. Because of this, the execution of a virtual test is not restricted to a local workstation anymore, but can be "outsourced" to other platforms or even server farms – a significant plus particularly for international companies with designers remotely working from different places.

ATE Interface: The test program initiates the execution of any operation by a function call of the ATE Interface. After the function call has been applied, this information is then forwarded to the DUT. Since now two DUTs could be available (the actual DUT or a virtual one), a mechanism to select the target we want to test is needed. This requires changes in the ATE Interface. Moreover, as we want to hide all the details from the user (he/she should not need to know on what DUT the test program is executed), each instrument has to be re-implemented in order to (1) keep the interfaces to the user the same and (2) generate the proper packages for both, the hardware tester and the virtual tester. Since existing tester software is actually proprietary, this required the complete re-implementation of the ATE interface to realize these changes.

However, once such an implementation is available, further virtual testers (i.e. targets) can be integrated very easily. Moreover, the costs for the realization are a non-recurring investment for each particular test system, i.e. several projects can be realized based on it after the system is available.

Overall, this leads to a virtual ATE which enables the virtual test methodology proposed in this paper. In fact, every ASIC design for which an ESL model is available can now be tested using the virtual description. By this, errors in the test program can be checked early so that, eventually, a correct test program is ready for application right after first-silicon of the ASIC is available.

IV. Industrial Case Study

The virtual test methodology proposed in this work has completely been implemented as described above and evaluated within an industrial environment. In the following, we demonstrate the application of the resulting solution by means of a representative case study. To this end, we consider the test program development for an ASIC designed within *Infineon Technologies*. In the following, the ASIC as well as the correspondingly developed test program are briefly reviewed. Afterwards, it is shown how the proposed virtual test methodology helped in checking the test program prior to first silicon[3].

A. Considered ASIC and Test Program

In this case study, the design of an ASIC is considered which is equipped with a *Serial Peripheral Interface* (SPI) to be tested. The interface is supposed to be used for the communication with the internal registers of the chip. The chip has a fixed number of internal registers which can all be addressed using a digital interface. Not all registers are writable. Following the design flow reviewed before by means of Fig. 1, an ESL model of this design has been implemented in SystemC-AMS (in version SystemC-AMS 2.0 [17]) using the system level design environment *COSIDE* [18]. This special integrated design environment is build up on top of the *Eclipse* framework [19]. Overall, this yielded a description which was used as virtual DUT.

In parallel to the design process, the development of the test program started. One aspect of this test program was the execution of a memory test. To this end, an implementation of the *March-X* algorithm has been applied which is a member of the widely known *March algorithm family* for memory testing (see e.g. [20] for details). More precisely, the following four steps have been conducted:

- $\Updownarrow$ $(w0)$: All registers are set to "0".
- $\Uparrow$ $(r0, w1)$: All registers are checked for their value and are set to "1".
- $\Downarrow$ $(r1, w0)$: All registers are checked again and are set to "0" again.
- $\Updownarrow$ $(r0)$: All registers are checked again for their correct value.

According to [20], applying this March-X algorithm is capable of covering fault models like stuck-at fault, address decoding faults, as well as transition faults.

B. Obtained Results

The ASIC reviewed above has been tested with the test program using both, the original ATE (after a first silicon realization was available, i.e. rather late in the design process) and the virtual ATE (after an ESL model was available, i.e. rather early in the design process). In the former case, the test has been executed based on a *Teradyne UltraFLEX* test system [21]. In the latter case, the virtual ATE as described in Section III has been executed on a local workstation with an Intel Xeon CPU and 16 GB of RAM running Microsoft Windows.

The correspondingly obtained user outputs are provided in Fig- 3. By applying the original flow (e.g. by utilizing the original tester), an error could be unveiled as reported in lines 9-12 of the left-hand-side of Fig. 3. The test program reported that one register (namely $0x258$) failed the memory test. In order to test the applicability of our virtual test methodology, we have also executed this test by applying the test program together with the virtual ATE. As can be seen on the right-hand-side of Fig. 3, the error could also be unveiled by applying our methodology which ended up in finding the same error. Finally the error which could be unveiled by both had been caused by the fact that not all register were writeable.

Following the original test flow, the execution of this test lasted some few seconds. In contrast, using the virtual test flow, approx. 2 hours were needed. This was expected as, of course, the execution on real hardware is faster than its corresponding simulation on software. Nevertheless, the proposed virtual test flow clearly outperforms the original flow. In fact, although a longer run-time is needed, the virtual test can be executed before first silicon is available – usually months before a first test was possible using the original test flow. This provides a clear benefit as the error can be detected and fixed significantly earlier in the design process. In contrast, detecting the error in the test program when first-silicon is available would have caused a debugging loop at a time, where actually the mass-production and deployment of the chip is due – yielding a significant delay in the production cycle.

Besides that, the obtained results clearly show that the proposed virtual test methodology is completely generic and can be conducted in a fully black-box fashion. In fact, exactly the same test program has been applied in both cases and, as confirmed by Fig. 3, exactly the same outputs were generated (except for the required run-time). This demonstrates that the user actually does not need to know whether he/she works with an actual DUT or a virtual DUT. Finally, the virtual solution proposed here offers further advantages because of its broader applicability. Since ATE hardware is very expensive, usually very restricted access is granted to the user. In contrast, the proposed virtual ATE gives the users unlimited accesses to multiple tester platforms offering the same behavior as an original ATE tester. Even remote access (important particularly for international companies with testers working from different places) is possible. These benefits have been confirmed on several further design projects at Infineon Technologies as well.

V. Conclusion

In this paper, we have proposed a virtual methodology for the development of test programs for industrial designs. To this end, we revisited all major steps conducted by an ATE thus far, and developed corresponding virtual, i.e. software,

[3]Note that, due to IP reasons, not all details of this case study (in particular of the considered ASIC) can be revealed. Those information are, however, not needed to demonstrate the benefits of the proposed methodology.

```
Site Number:
  0, 1, 2, 3

Device#: 1

tfMemoryTest1
Total test duration: 6918 s
Individual test duration:
Reset:      1259.44 s, Failed Registers: (None)
Zero:       1883.15 s, Failed Registers: 0x258
Mask:       1889.29 s, Failed Registers: 0x258
SecondZero: 1886.12 s, Failed Registers: 0x258

Number   Site   Test Name
<MemoryTest>
16000    0    MEM_ETT
16010    0    MEM_Reset
16011    0    MEM_Zero
16012    0    MEM_Mask
16014    0    MEM_SecondZero

Site Failed tests/Executed tests
----------------------------------------------
  0        3        5
Site     Sort      Bin
----------------------------------------------
  0        1        1
```

```
Site Number:
  0, 1, 2, 3

Device#: 1

tfMemoryTest1
Total test duration: 3.48 s
Individual test duration:
Reset:      1.51 s, Failed Registers: (None)
Zero:       0.72 s, Failed Registers: 0x258
Mask:       0.74 s, Failed Registers: 0x258
SecondZero: 0.52 s, Failed Registers: 0x258

Number   Site   Test Name
<MemoryTest>
16000    0    MEM_ETT
16010    0    MEM_Reset
16011    0    MEM_Zero
16012    0    MEM_Mask
16014    0    MEM_SecondZero

Site Failed tests/Executed tests
----------------------------------------------
  0        3        5
Site     Sort      Bin
----------------------------------------------
  0        1        1
```

Fig. 3: Test reports generated by both, the virtual ATE and the original ATE.

solutions for them. In contrast to the established design flow (as summarized in Fig. 1), this allows to check test programs even before first silicon is available and overcomes a serious bottleneck in today's industrial flows. A case study within Infineon Technologies confirmed the benefits of the proposed methodology. Testers were able to conduct the required tests before first silicon, i.e. months earlier than with the original test flow. This significantly reduced the pressure caused by time-to-market objectives.

ACKNOWLEDGEMENTS

The authors would like to thank K. Dominizi, E. Dorfer, H. Eichinger, S. Kampfer, G. Krebelder, B. Mariacher, O. Pfabigan and R. Reiterer for making this contribution possible.

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

REFERENCES

[1] G. Martin, B. Brian, and P. Andrew, *ESL Design and Verification: A Prescription for Electronic System Level Methodology*. San Francisco, USA: Morgan Kaufmann Publishers, 2007.

[2] "IEEE Standard SystemC Language Reference Manual," *IEEE Std 1666-2005*, pp. 1–423, 2006.

[3] "IEEE Standard VHDL Language Reference Manual," *IEEE Std 1076-2008*, pp. 1–640, 2008.

[4] "IEEE Standard Verilog Hardware Description Language," *IEEE Std 1364-2001*, pp. 1–856, 2001.

[5] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic Model Checking Without BDDs," in *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*, London, UK, 1999.

[6] P. Gonzalez-de Aledo, N. Przigoda, R. Wille, R. Drechsler, and P. Sanchez, "Towards a Verification Flow Across Abstraction Levels Verifying Implementations Against Their Formal Specification," *Transactions on CAD of Integrated Circuits and Systems*, vol. 36, no. 3, 2017.

[7] T. Larrabee, "Test pattern generation using boolean satisfiability," *Transactions on CAD of Integrated Circuits and Systems*, vol. 11, no. 1, Jan 1992.

[8] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved SAT-based ATPG: More Constraints, Better Compaction," in *Proceedings of the Int'l Conf. on CAD*, San Jose, USA, 2013.

[9] M. Burns and G. W. Roberts, *An Introduction to Mixes-Signal IC Test and Measurement*. New York City, USA: Oxford University Press, 2001.

[10] M. L. Bushnell and V. D. Agrawal, *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. New York City, USA: Kluwer Academic, 2002.

[11] J. O'Doherty, "A Virtual Test Approach to Mixed Signal Test Development," in *Colloquium on Systems on a Chip*, London, UK, 1999.

[12] T. Austin, "Creating a Mixed-Sigal Simulation Capability for Concurrent IC Design and Test Program Development," in *Proceedings of the Int'l Test Conf.*, Baltimore, USA, 1993.

[13] G. Krampl, M. Rona, and H. Tauber, "Test Setup Simulation - a High-Performance VHDL-based Virtual Test Solution Meeting Industrial Requirements," in *Proceedings of the Int'l Test Conf.*, Baltimore, USA, 2002.

[14] A. I. Voinea and S. Kampfer, "Rapid Prototyping and Test before Silicon of Integrated Pressure Sensors," in *Proceedings of the Int'l Test Conf.*, Anaheim, USA, 2015.

[15] "HDL Simulation Environment," https://www.mentor.com/products/fv/modelsim/, Mentor Graphics, (accessed: 01.05.2019).

[16] Teradyne: Spectrum hs – functional test platform. http://www.teradyne.com/products/defense-aerospace/spectrum-hs. (accessed: 01.05.2019).

[17] "Standard SystemC AMS extensions 2.0: Language Reference Manual," Accellera, 2013.

[18] Coseda Technologies: System Level Design Environment COSIDE. , note = (accessed: 01.05.2019).

[19] Eclipse Foundation: Eclipse. https://eclipse.org/. (accessed: 01.05.2019).

[20] W.-B. Jone, *Memory Testing*. Cincinnati, OH, USA: University of Cinncinati, 2008.

[21] Teradyne: Ultraflex test system. http://www.teradyne.com/products/semiconductor-test/ultraflex. (accessed: 01.05.2019).