2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Towards a UVM-based Solution for Mixed-signal Verification

Alexander W. Rath, Sebastian Simon, Volkan Esen, Wolfgang Ecker
Infineon Technologies AG
Email: *Firstname.Lastname*@infineon.com

***Abstract*—The Universal Verification Methodology (UVM) has become a de facto standard in today's functional verification of digital designs. However, it is rarely used for the verification of Designs Under Test containing Real Number Models. This paper presents a new technique using UVM that can be used in order to compare models of analog circuitry on different levels of abstraction. In order to create pass-fail-criteria, it makes use of a similarity metric called Earth Mover's Distance. The presented technique enables us to ensure that Real Number Models used in chip projects match the transistor level circuitry during the whole life cycle of the project.**

I. INTRODUCTION

In today's IC designs more and more parts of the analog implementation are shifted to the digital domain, since digital circuits scale better with new technologies. This trend leads to mixed signals designs. Their analog and digital parts interface with each other as well as with the outside world.

The functional verification of the analog parts is different compared to the verification of the digital parts:

- Digital parts are functionally verified on register transfer level (RTL). Very sophisticated transaction-based methodologies like OVM [1] or UVM [2] are used in order to accomplish this task. The key concepts of these methodologies are the generation of constrained-random stimulus [3], automated checking mechanisms [4] and the collection of functional coverage [5].
- The analog parts of mixed signal designs are usually verified on SPICE level by using network simulators [6]. This approach covers mainly the verification of electrical parameters, e. g. input resistance and amplification. However, it is also used to verify the functional behavior of the block.

In the verification of the whole chip (chip level verification), where the system-level behavior as well as the interconnectivity of the blocks are to be checked, the detailedness of the SPICE models is often not required. Also, they slow down the simulation speed drastically.

In consequence, it is a common practice not to use the SPICE models in chip level simulations. Instead, so called real number models (RNM) are used [7], [8]. They purely reflect the functional behavior of the analog parts and are developed by using a hardware description language, e. g. VHDL, Verilog or SystemVerilog. The advantage of this approach is that a regular event driven simulator can be used in order to perform the chip level verification.

As the specification of the SPICE modules changes often during the project development cycle, due to new findings, it becomes a challenge to keep the aforementioned RNMs consistent to the their SPICE counterparts.

A big portion of this challenge is to verify the consistency of RNMs and their SPICE counterparts, because the evaluation is usually done by visual inspection of simulator wave forms over a limited set of test stimuli. On one hand, this approach is error prone and on the other hand very time consuming.

In this paper, we present a verification approach which enables automatic reevaluation of the consistency between the SPICE models and the RNMs, while allowing a much broader coverage of tested functionality.

This approach is based on an extension of the UVM, which is state of the art [9] in automated digital verification.

In the following chapters we present an outline of the approach and how it is mapped to the UVM. Furthermore, we describe the application of this approach to a typical example and provide an overview of experimental results. In connection to that, an analysis with regard to related work in this area is given, followed by a conclusion and an outline of steps that are to be addressed next.

II. APPROACH

In order to ensure consistency between the RNM and its SPICE counterpart, it is mandatory to stimulate both models with the same stimuli and automatically compare the results of both models for equality.

In principal, UVM offers these mandatory features. However, it is originally intended for digital stimulus generation and the comparison of digital signal sequences. Our purpose, however, requires the generation of a wide range of analog stimuli and the comparison of analog signal sequences.

While digital behavior represents a sequence of binary values at specific points in time, analog behavior represents a continuous progression of arbitrary values, i.e. a function $t \mapsto f(t)$. Hence, it is necessary to provide a framework which on hand can create stimulus in such a way and on the other hand, provides methods to compare such kind of behavior.

In addition to this an RNM is typically more abstract and hence, creates a simplified behavior compared to the SPICE counterpart. Therefore, the framework also needs to provide mechanisms, which allow tolerances in the comparison. The acceptable tolerances are to be defined upfront in order to enable a clear pass/fail criterion.

In the following sections, we describe how we enhance the UVM framework in order to support these requirements. Our enhancement to UVM, we call A-UVM (analog UVM).

A. Transactions

The core elements in UVM-based testbenches are the transactions. In the following, we shortly explain that claim and the consequences, it implies for testbenches. Furthermore, we extend the concept of transactions towards the analog world.

1) Transactions in UVM

Not only in a UVM context, a transaction is a set of parameters, which describes a certain protocol in an abstract manner above the signal level. For example, a transaction for a simple serial protocol may contain a parameter ”address” and another parameter ”data”. However, it will not contain information about how exactly the transaction will look like on pin level. Thus, the same transaction could potentially be used to abstractly describe a simple parallel protocol. The motivation of modeling protocols using transactions is that a verification engineer is usually not interested in the signal level behavior of the design. For example, if it is to be checked that a certain register in the DUT holds the correct value, only the address and data information of the respective communication is of interest. There is no reason for carrying the information about what exactly happened on pin level. However, in order to communicate with a DUT, the transaction has to be translated from transaction level to pin level and vice versa. In a UVM testbench so-called drivers and monitors are used to accomplish this task. Together, a driver and a monitor form an agent, which is also called a verification component (VC). Hence, the transactions that are used to communicate with the DUT determine the structure of the testbench.

In UVM, the code to describe the example transaction mentioned would look like this:

```
class my_item extends uvm_sequence_item;
  int address;
  int data;
  `uvm_object_utils_begin(my_item)
    `uvm_field_int(address, UVM_ALL_ON)
    `uvm_field_int(data,    UVM_ALL_ON)
  `uvm_object_utils_end
endclass
```

2) Analog Transactions

Analog signals are different compared to digital signals, as their co-domain is practically unlimited. That allows single analog signals to adopt different shapes, whereas a single digital signal is always of rectangular shape. Despite this, it is possible to classify the shape of an analog signal. For example, an analog signal can be of a linear, harmonic or cubic spline shape or of any other shape as well. Obviously, in order to precisely describe an analog signal, it is not sufficient to simply name its shape. Additional parameters are required. For example, in order to describe a linearly shaped signal, its slope as well as one value at a certain point in time are to be specified.

In A-UVM, we identify the term ”shape” with the term ”protocol” known from purely digital interfaces. The term ”transaction” stands for a data structure which contains the parameters needed to specify an analog signal.

For example, a signal with a sinusoidal shape would be described by the two real-valued parameters ”amplitude” and ”frequency”.

However, in our approach we found it necessary to add some meta data to the transactions. Thus, we separated the transaction parameters from the sequence item, by defining a new class a_uvm_data_structure. This class serves as a base class for a new class containing the analog transaction parameters.

```
class a_uvm_sine_data_str extends a_uvm_data_structure;
  rand a_uvm_rand_real ampl;
  rand a_uvm_rand_real freq;
  ‘uvm_object_utils_begin(sine)
    ‘uvm_field_real(ampl, UVM_ALL_ON)
    ‘uvm_field_real(freq, UVM_ALL_ON)
  ‘uvm_object_utils_end
endclass
```

Note that we had to introduce a new real number class a_uvm_rand_real [10] since the SystemVerilog standard only supports randomization of integral variables, that is, variables that contain sets of bits [11], [12]. Therefore, a type conversion from real to bit (rtb) was required. This class furthermore offers the possibility to select a distribution type that is used during randomization (e. g. uniform, exponential, Gaussian).

Now, such an analog sequence item can be issued in a way that is almost equivalent to the well known UVM way of using the `` `uvm_do() `` macro family:

```
a_uvm_sequence_item sine_item;
‘uvm_create(sine_item)
sine_item.data_str = a_uvm_sine_data_str::type_id::create(”sine”);
‘uvm_rand_send_with(sine_item, {
          data_str.freq.minv == rtb(190.0e3);
          data_str.freq.maxv == rtb(210.0e3);
          data_str.freq.dist_type == UNIFORM;
          data_str.ampl.minv == rtb(−5.5);
          data_str.ampl.maxv == rtb(−4.5);
          data_str.ampl.dist_type == GAUSSIAN;
})
```

The sequence item itself references the data structure and contains three meta data fields (algorithm name, duration, sample rate), which we explain in the next sections.

B. Generating Constrained-random Analog Stimulus

In UVM-based testbenches, drivers are used to transform transactions to signal level activity. In this section we show how A-UVM accomplishes this task regarding the analog transactions defined in the previous section.

In purely digital environments, a protocol is assigned to the digital driver using an FSM. According to the previous paragraph, a signal shape must be assigned to the analog driver. However, this cannot be done using an FSM. Instead a numerical algorithm must be used.

A-UVM uses a predefined interface for the communication between a generic driver and the algorithms. This interface allows new – usually project-specific – algorithms to be plugged in. It also allows to exchange the algorithm to be used during runtime. This plug-in mechanism is realized using the so-called ”strategy pattern” from [13]. Regarding algorithms, A-UVM is not restricted to SystemVerilog. Algorithms written in C, Matlab or other languages can be plugged in as well.

The interface that is to be provided by every algorithm consists of the following methods. We explain them in the following paragraphs.

```
pure virtual function void pre_process(a_uvm_data_structure data_str);
...
pure virtual function real get_real(real x);
...
virtual function void post_process();
```

When the driver receives a potentially randomized transaction, it reads the meta field algorithm_name. Then the driver selects this algorithms from its data base and passes the field data_str to the algorithm by calling its method pre_process. This algorithm can use this in order to open a connection to external tools, e.g. to Matlab, if required.

After that the driver starts to call get_real repeatedly. The interval in which the driver calls this method is determined by the meta field sample_rate in the transaction. The argument x of the method represents the time elapsed since the start of the transaction. This information is used by the algorithm in order to compute the actual signal value. It is returned by the method get_real. The whole process lasts until the time specified by the meta filed duration in the transaction is elapsed. After that, the driver calls post_process. Upon this call,

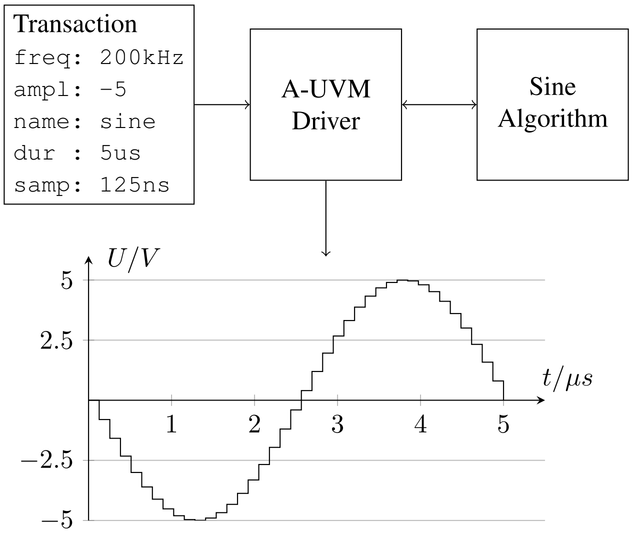

Figure 1. Process of driving an analog transaction onto a signureal in A-UVM resulting in a sampled sine waveform.

the algorithm can perform finalization tasks, e.g. closing the connection to an external tool. Now, the A-UVM driver is ready for the next transaction, which can be processed by the same or another algorithm.

Fig. 1 visualizes the whole process at the example of a sine wave. However, A-UVM is not restricted to sine waves. It features algorithms for FOURIER synthesis, cubic spline interpolation, ramps, jumps and can be easily expanded by user-defined algorithms.

C. Monitoring Analog Behavior

In this section we show, how we extend the concepts presented in the previous sections towards monitoring of analog signals.

In principle, the definitions from the previous sections can be used directly for monitoring as well: An analog monitor must be able to extract the parameters of a signal of a given shape. In order to do so, the monitor uses a – potentially project dependent – algorithm reflecting the signal shape. Because of the potential project dependency of the algorithm, we separated the monitor from the monitoring algorithm it deploys. This allows us to keep all the management required for monitoring in our library. Also, it allows the verification engineer to exchange the monitoring algorithm during run time. This is useful, if the shapes that are to be monitored vary over time because of the DUT being in different states. In the following subsections we present the plug-in approach in more detail as well as some complications that need be taken into account which we discuss in the following sections. Since the complications have an impact on the plug-in mechanism, we go for them first.

1) Triggering

When stimulating a DUT's interface, the driver is not responsible of determining the point in time, when exactly a transaction is supposed to start. Instead, it just starts to drive the DUT's interface once it receives the transaction from the sequencer. The sequencer in turn is controlled by higher testbench facilities, e.g. by a test sequence.

When monitoring a DUT's interface, this circumstance is substantially different. There is no testbench facility that can tell the monitor when to start exactly. Instead, the monitor has to determine the start of a transaction by recognizing a characteristic activity in the DUT's output.

For analog signals, typical start activities could be e.g.

- discontinuities,
- discontinuities in the first derivative,
- changes in frequency or
- crossing a certain value.

Of course, other characteristics are possible as well.

In order to enable the monitor to start the construction of a transaction at one of the aforementioned activities, we added the concept of triggers to A-UVM. A trigger is an object that raises an event upon one of the

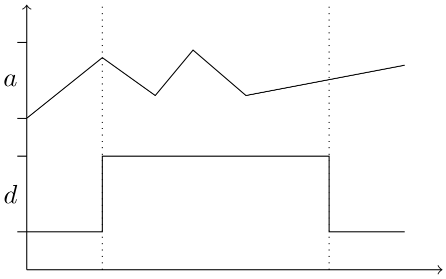

Figure 2. Arbitrary analog signal a and arbitrary digital signal d. If the monitor algorithm is supposed to determine the greatest value of a within the interval where d is high, it needs the help of a start and a stop trigger, in order to determine start and end time.

aforementioned activities. In A-UVM, we defined triggers covering these activities. However, additional user-defined triggers can be plugged in via a callback mechanism. Hence, the approach provides the flexibility required to be useful across different projects.

In the previous paragraph, we focused on a start activity *within* the signal to be monitored. However, there can be cases, where the start is actually not an activity within the signal to be monitored. The start activity could be an activity in another signal – analog or digital. Hence, we designed A-UVM in such a way that the trigger can be sensitive not only to the signal to be monitored (see fig. 2). Furthermore, we designed it in a way, such that triggers can be combined. This way, triggers can form logic interconnections.

2) Single-threaded vs. Multi-threaded

The monitor in A-UVM knows two operating modes:

- Single threaded, i. e. serial operation mode and
- Multi threaded, i. e. parallel operation mode

In the example of fig. 2, a rising edge and a falling edge of d can never occur at the same point in time. Therefore, the transactions in this example are not interleaved. Hence, the monitor must operate in sequential mode, i. e. only one instance of the algorithm is active at the same time.

However, in the example fig. 3, the situation is different. Let us assume that the characteristic to be monitored is the time between a discontinuity and a zero. The actual time T_i between a discontinuity and a zero is measured by the monitoring algorithm. The discontinuities are detected by a start trigger and the zero is detected by a stop trigger (or – alternatively – by the algorithm itself). In this example, the monitor has to spawn three algorithms, because there are three discontinuities before the zero. Therefore, the monitor must operate in parallel mode. If, instead, the monitor was operating in sequential mode, the transactions belonging to the intervals T_2 and T_3 in fig. 3 would not be generated, since the monitor would be blocked by determining T_1.

3) Plug-in Approach for Monitoring

The A-UVM monitor communicates with one or more attached monitoring algorithms via a predefined API. A verification engineer who designs a new monitoring algorithm has to create a class whose methods comply

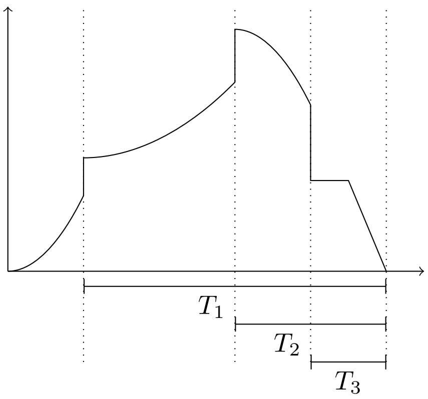

Figure 3. Arbitrary analog signal with discontinuities. If the purpose of our monitor is to measure the time between the discontinuities and the zero, it must operate in parallel mode in order to capture all three times.

with this API. Thus, the API serves as an abstraction layer. In this subsection we present the plug-in API in more detail.

All monitoring algorithms have to extend the virtual class `a_uvm_monitor_algorithm` which is part of our library. This class defines three virtual methods which are to be overridden by the concrete monitor algorithms:

```
pure virtual task start(a_uvm_sequence_item trans);
```

This method is called by the generic monitor, once a transaction is started. This happens once a start trigger attached to the monitor has encountered a respective event. Since the method is pure virtual, the user has to provide a project dependent implementation, which monitors the transaction. The monitoring algorithm stores the results it extracted in a data structure that we call `a_uvm_data_structure`. The concrete layout of this data structure depends on the monitoring algorithm. Thus, we provide it as a virtual class as well. After having filled the data structure, the monitoring algorithm attaches it to the field `data_str` in the argument `trans`.

4) The Generic Monitor

In this subsection we present the generic monitor in more detail. Its task is to perform all the required management for the communication between triggers and the actual monitoring algorithm. The class is a library element. Hence, it is not required to override it.

However, it can be configured dynamically through a small set of methods, which we will present within this subsection as well. Since it is not required to override these methods, they are non-virtual.

```
function void register_monitor_algorithm (a_uvm_monitor_algorithm algorithm);
```

Using this method, algorithms according to the previous subsection can be registered. Being able to register more than one algorithm is useful, since it is sometimes desired to be able to monitor different signal shapes within one simulation.

```
function void set_monitor_algorithm (string algorithm_name);
```

This method selects one of the registered algorithms. This is the algorithm used by the generic monitor from now on.

```
function void set_monitor_operation_mode (a_uvm_operation_mode_type operation_mode);
```

Through this function one can select whether the monitor should run in sequential or parallel mode. Additionally, one can completely shut down the monitor through this method.

```
function void set_start_trigger (a_uvm_monitor_trigger trigger);
```

Using this method, the triggers described above can be registered. Based on the configurations set through the aforementioned methods, the A-UVM monitor provides all the required management. This includes

- Reacting on triggers
- Starting of algorithms
- Pushing monitored transaction through a TLM port to higher testbench facilities

D. Checking Analog Transactions

In UVM-based testbenches, the transactions produced by monitors and reference models are to be compared in order to determine, whether the DUT behaves correctly.

Comparing transactions in UVM is done by comparing their fields bit-wise. However, analog transactions have real-valued fields. Due the fact that real value operations are not exact [14], a bit-wise comparison of the fields of an analog transaction will almost never lead to accordance.

For this reason, we introduced a technique for the automated comparison of analog behavior based on a metric which measures the distance between two feature vectors—in our case represented by A-UVM transactions [15]. The algorithm is based on the earth mover's distance, which is widely used in content-based image retrieval [16]. It computes the minimum costs for turning one transaction into another by solving a transportation problem, which in turn can be solved by linear optimization. This algorithm is implemented in SystemVerilog via DPI-C following the UVM standard and enables us to quantify a degree of similarity between an analog waveform and a reference signal.

The introduced metric yields a value d_{EM} within the interval $[0, 1]$ where 0 symbolizes complete dissimilarity and 1 a full match:

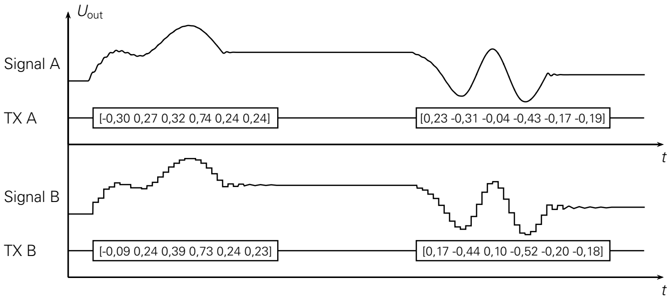

Figure 4. Extracted transactions (TX) from two analog signals A and B by equidistant sampling

- $d_{\text{EM}}(\boldsymbol{x}, \boldsymbol{y}) = 1$: Feature vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are identical.
- $d_{\text{EM}}(\boldsymbol{x}, \boldsymbol{y}) = 0$: Feature vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ behave completely contrary.
- $0 < d_{\text{EM}}(\boldsymbol{x}, \boldsymbol{y}) < 1$: Feature vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are neither identical nor behave completely contrary. However, the value for d_{EM} quantifies their degree of similarity.

Fig. 4 shows an example of two analog signals where equidistant sampling is chosen as extraction method. The number of monitored samples per transaction is set to $n = 6$ (in the following also referred as number of data points). The extracted transactions are being compared pairwise in order to determine a distance d_{EM} for each pair.

In a first step, a lower bound for the expected similarity d_{EM} has to be defined. If d_{EM} then falls below this bound the corresponding transactions can be located and examined in order to identify the mismatch between the flagged waveforms. This test procedure can be run completely automated after setting up test cases and determining appropriate extraction methods.

III. RELATED WORK

UVM [2] is the emerging de facto standard for creating reusable testbenches and verification environments. Released by Accellera, this standard defines a class library, which allows verification engineers to build verification components (VCs) and environments in a standardized way. Further, the UVM class library provides a callback mechanism, which enables VCs and system models to communicate via TLM (Transaction Level Modeling) [17].

For the analog domain, no such abstract communication technique is available. However, several different approaches to extend modern hardware, verification and system description languages with the ability to describe analog behavior have been developed; the newest being SystemC-AMS, presented in [18]. SystemC-AMS allows modeling engineers to describe analog behavior in frequency and time domain.

The drawback is that no verification library, that is such sophisticated as UVM, exists for SystemC or SystemC-AMS.

Another new approach is UVM-MS presented in [19] and [20]. However, this approach focuses mainly on the direct stimulation of the pins of the DUT using UVM and an additional Verilog-AMS layer.

All these newer approaches for the analog domain enable the verification of AMS models. The difference between AMS models and the aforementioned RNMs is that AMS models aim more at a higher level of electrical accuracy that is often not required for chip level verification, since in a chip level verification the over all functionalities rather than the electrical parameters are of interest. In consequence, none of the approaches mentioned in this section fits to our verification problem described in section I.

IV. APPLICATION AND DISCUSSION

In this section we show an application of the approach presented in the previous paragraphs.

The application is a voltage regulator circuit for which an RNM written in SystemVerilog has been developed, in order to speed up chip level simulation. Since jump stimuli cause the most significant output for regulators, we have built a testbench (see fig. 5) that stimulates both regulators with jumps of random height and evaluates the responses of both models for consistency. The step responses upon a unit jump of both models are shown in fig. 6.

The testbench's driver produces jumps. The height of one jump is the only parameter of the incoming transaction and is randomized by the test sequence. The monitors calculate the frequency spectra of the model's

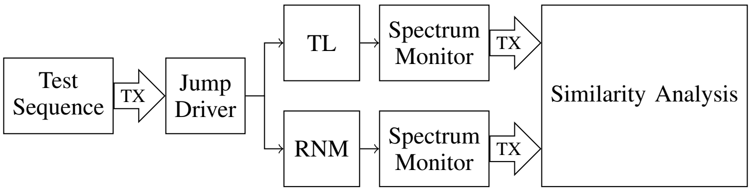

Figure 5. The testbench used in order to compare two different abstraction levels of a voltage regulator

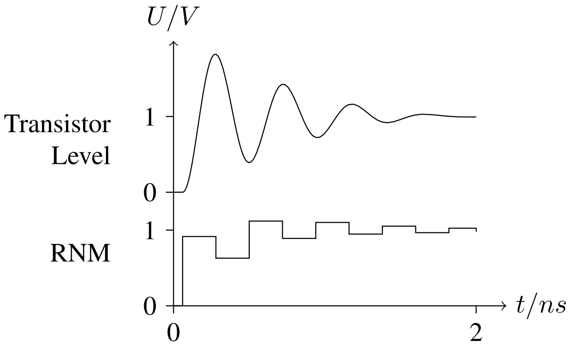

Figure 6. Step responses of the two voltage regulator models. The frequency of the transient oscillation is the same for both models. That indicates, that the frequency spectra of both models can be correlated, in order to evaluate the consistency. If the oscillation frequency of the transistor level circuit was changing due to a change in its design, the similarity of the produced waveforms would become much smaller, indicating that the RNM is to be updated.

outputs within an interval of 2 ns. One transaction produced by a monitor carries the frequency spectrum of one step response, i.e. a list of complex values. After that, the comparison component of the testbench successively calculates the similarity of each pair of transactions.

Our test sequence produces 1000 random jumps in a row. The first version of the RNM simulated together with its SPICE counterpart led to a similarity coefficient greater than 0.89 for every produced frequency spectrum pair. We ran this test sequence in a nighty regression. After a design change in the transistor level regulator, the oscillation frequency was reduced by a factor of 2. Since the RNM was not updated, the similarity dropped down to a value smaller than 0.24 for every transaction pair. This was a clear indication that an update of the RNM is required. After updating the RNM, the similarity went up to 0.9 again.

The effort for constructing the testbench was about one week. In contrast, the effort for building a testbench relying on directed tests and manual wave form checking is slightly smaller. However, such a testbench can not be run in nightly regressions and the effort that is needed in order to check the consistency between the two models manually over and over again during the project exceeds the effort spent for our approach by far. Furthermore, the A-UVM-based approach presented in this paper features randomization. Thus, it covers corner cases that are easily forgotten in a directed testbench.

V. CONCLUSION AND OUTLOOK

In this paper we introduced a strategy for verifying analog behavior and highlighted its key features. The technique tackles the necessity of being able to compare models with analog behavior on different levels of abstractions.

Our future work will focus on the extensions of the presented methodology, regarding usability and flexibility. The goal is to provide a UVM-based building box that covers the need of verification engineers to simulate and verify designs containing real number models. This building box shall include methods and techniques for driving, monitoring and checking of analog signals, as well as for coverage collection and reference modeling.

REFERENCES

[1] "OVM User Guide – Version 2.1.2," June 2011. [Online]. Available: www.verificationacademy.com
[2] "Universal Verification Methodology (UVM) 1.1 User's Guide," May 2011. [Online]. Available: www.uvmworld.org
[3] N. Kitchen and A. Kuehlmann, "Stimulus generation for constrained random simulation," in *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, ser. ICCAD '07. Piscataway, NJ, USA: IEEE Press, 2007, pp. 258–265. [Online]. Available: http://dl.acm.org/citation.cfm?id=1326073.1326127

[4] G. Allan, “Architectural considerations of scoreboard design,” in *Proceedings of the 2012 DVCON international conference*, ser. DVCON ’12, 2012. [Online]. Available: http://events.dvcon.org/2012/proceedings/papers/01P_7.pdf

[5] H. Carter and S. Hemmady, *Metric Driven Design Verification: An Engineer’s and Executive’s Guide to First Pass Success*. Springer, 2010. [Online]. Available: http://books.google.de/books?id=sAWtcQAACAAJ

[6] P. Li, L. Silveira, and P. Feldmann, *Simulation and Verification of Electronic and Biological Systems*. Springer, 2011. [Online]. Available: http://books.google.de/books?id=N48SiipG8LkC

[7] A. Elzeftawi, “CDNLive! – Real Number Model Development and Application in Mixed-Signal SoC Verification,” April 2012. [Online]. Available: http://www.cadence.com/Community/blogs/ms/archive/2012/04/09/cdnlive-real-number-model-development-and-application-in-mixed-signal-soc-verification.aspx

[8] W. Hartong and S. Cranston, “Real Valued Modeling for Mixed Signal Simulation,” January 2009. [Online]. Available: http://www.cadence.com/rl/Resources/application_notes/real_number_appNote.pdf

[9] T. Poikela, J. Plosila, T. Westerlund, J. Buytaert, M. Campbell, X. Llopart, R. Plackett, K. Wyllie, M. van Beuzekom, V. Gromov, R. Kluit, F. Zappon, V. Zivkovic, C. Brezina, K. Desch, X. Fang, and A. Kruth, “Architectural modeling of pixel readout chips velopix and timepix3,” *Journal of Instrumentation*, vol. 7, no. 01, p. C01093, 2012. [Online]. Available: http://iopscience.iop.org/1748-0221/7/01/C01093

[10] S. Simon, G. Pelz, and L. Maurer, “Accelerating Coverage Collection for Mixed-Signal Systems in a UVM Environment,” in *Forum on Specification and Design Languages (FDL)*, September 2015, p. 31.

[11] C. Spear, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*, 3rd ed. Springer, February 2012.

[12] *IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, IEEE Computer Society and the IEEE Standards Association Corporate Advisory Group, February 2013.

[13] E. Gamma, *Design Patterns: Elements of Reusable Object-Oriented Software*, ser. Addison-Wesley Professional Computing Series. Addison-Wesley, 1995. [Online]. Available: http://books.google.de/books?id=6oHuKQe3TjQC

[14] *The Art Of Computer Programming, Volume 2: Seminumerical Algorithms, 3/E*. Pearson Education, 1998.

[15] S. Simon, A. W. Rath, V. Esen, and W. Ecker, “Automated Comparison of Analog Behavior in a UVM Environment,” in *Design and Verification Conference*, March 2014. [Online]. Available: http://events.dvcon.org/2014/proceedings/papers/05_1.pdf

[16] S. Cohen, “Finding Color and Shape Patterns in Images,” Ph.D. dissertation, Stanford University, May 1999. [Online]. Available: http://i.stanford.edu/pub/cstr/reports/cs/tr/99/1620/CS-TR-99-1620.pdf

[17] M. Glasser and J. Bergeron, “Tlm-2.0 in systemverilog,” in *Proceedings of the 2011 DVCON international conference*, ser. DVCON ’11, 2011. [Online]. Available: http://events.dvcon.org/2011/proceedings/papers/04_2.pdf

[18] *OSCI SystemC-AMS extensions*, March 2010. [Online]. Available: www.systemc-ams.org

[19] N. Khan, Y. Kashai, and H. Fang, “Metric Driven Verification of Mixed-Signal Designs,” March 2011.

[20] N. Khan and Y. Kashai, “From Spec to Verification Closure: A Case Study of Applying UVM-MS for First Pass Success to a Complex Mixed-Signal SoC Design,” February 2012.