Generation of Constraint Random Transactions for Verification of Mixed-Signal Blocks

Alexander W. Rath*, Sebastian Simon*, Volkan Esen* and 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. It makes use of statistic metrics. 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 degree of detail of the SPICE models is often not required. Also, they slow down the simulation speed by magnitudes.

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 just 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 UVM is state of the art [9] in automated digital verification, it is desirable to perform the verification of such a chip-level using UVM as well. Thus, we have presented in [10], how analog stimulus can be generated and in [11], how analog responses can be checked using UVM. In this paper, we will present how the generation of analog sequences and transactions can be done efficiently with SystemVerilog and UVM.

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 verify a chip-level consisting of RTL models and RNMs, it is mandatory to stimulate it with analog and digital signals and automatically check its responses—digital as well as analog—for functional correctness.

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 the one hand can create stimulus in such a way and on the other hand, provides methods to compare such continuous behavior.

In this paper we will focus on the constrained random generation of analog transactions and sequences. Thus, in the following sections, we describe how we enhance the UVM framework in order to support this feature. 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 explain this statement 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 often 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;
  rand int address;
  rand 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 a transaction holding 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 sinus extends a_uvm_data_structure;
  real ampl;
  real freq;
  `uvm_object_utils_begin(sinus)
    `uvm_field_real(ampl, UVM_ALL_ON)
    `uvm_field_real(freq, UVM_ALL_ON)
  `uvm_object_utils_end
endclass
```

The transaction itself references the data structure.

```
class a_uvm_sequence_item extends
 uvm_sequence_item;
  //transactions parameters
  rand a_uvm_data_structure data_str;
  //meta data
  string algorithm_name;
  protected a_uvm_tlm_time
   duration;
  protected a_uvm_tlm_time
   sample_rate;

  `uvm_object_utils_begin
    (a_uvm_sequence_item)
    ...
  `uvm_object_utils_end
endclass
```

B. Generating analog stimulus from transactions

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 engraved into the digital driver using an FSM. According to the previous paragraph, a signal shape must be engraved into 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 – potentially 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 [12]. 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);
```

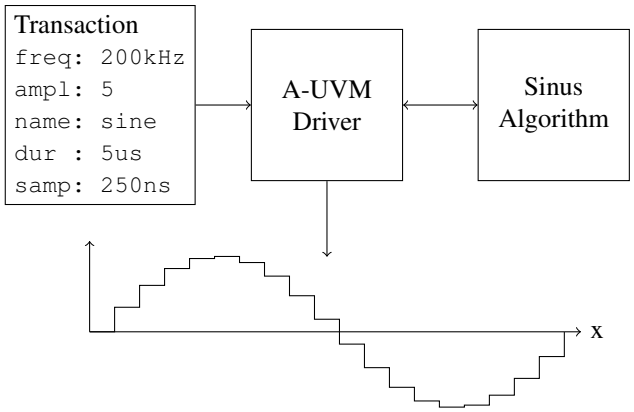

Figure 1. Process of driving an analog transaction onto a signal in A-UVM.

```
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, 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.

Figure 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 others.

C. Constrained random generation of transaction parameters

In the following sections, we show how digital transactions are generated using UVM and how analog transactions can be generated efficiently in A-UVM.

1) Generation of transactions in UVM: Transactions with integer parameters (see the example `my_item` in section II-A1) are usually generated by creating a new instance of it and then randomizing it, possibly with some constraints provided in a `with` clause. Since the `randomize()` method provides a return value which reports about the success, it is good practice to grab it in order to react on a possible randomization failure:

```
my_item tx = my_item::create("tx");
if(!tx.randomize() with {addr > 5;})
  `uvm_fatal("Rand fail", "...")
```

If the item is to be sent to a sequencer directly, the respective API of sequencer is to be called:

```
my_sequencer.send_request(tx);
my_sequencer.wait_for_item_done(tx);
```

In order to reduce the coding effort for this process, UVM provides the `` `uvm_do `` macro family:

```
`uvm_do_on_with(my_item, my_sequencer,
  {addr > 5;})
```

The macro shown in the example got widely adopted by verification engineers, as it is very compact and easily readable. Furthermore, it enables a declarative programming style for the creation of stimulus. This style is efficient and also provides some esthetic appeal.

2) Generation of analog transactions: Unfortunately, the method described in the previous section works for digital transactions only, but not for analog transactions. This is due to the fact that even the newest version of SystemVerilog does not allow to randomize `real` typed class members:

```
class my_class;
  rand real value; //Compile Error!
endclass
```

There are several options to work around this problem. We will discuss them in the following paragraphs.

One option is to simply declare the class without the `rand` keyword and to randomize the class member `value` via the detour of an integer variable `value_int` and a call to `$urandom`:

```
my_class m = new();
int unsigned value_int = $urandom % 500;
m.value = real'(value_int)/1000.0;
```

In this example, the the class member got randomized to a value smaller than 0.5. However, it has been done in a very procedural way, such that no constraints can be provided. This can be improved by using `std::randomize()` instead of `$urandom()`:

```
my_class m = new();
int unsigned value_int;
if(!std::randomize(value_int)
  with {value_int < 500;})
  `uvm_fatal("Rand fail", "...")
m.value = real'(value_int)/1000.0;
```

Now, the code is already much cleaner. However, there is still the detour via an integer variable and it is still not possible to specify constraints within the class `my_class`, as it has no truly random member.

It is possible to overcome those limitations by internalizing `value_int` to `my_class` and using the class's `post_randomize()` callback.

```
class my_class;
  rand int unsigned value_int;
  real value;

  function void post_randomize();
    value = real'(value_int)/1000.0;
  endfunction
endclass
//...
my_class m = new();
if(!m.randomize()
  with {value_int < 500;})
  `uvm_fatal("Rand fail", "...")
```

This solution is already feasible, as it already looks very similar to the digital example provided in the previous section. The last drawback is the fact that the division operation is very arbitrary. Although very intuitive, there is no formal reason to use a division. Any other function that uses two integers in order to produce a real-valued measure could be used instead. Expressed in mathematical terms it means that any function of the following form could be used:

$$r = f(a,b) \text{ with } (a,b) \in \mathbb{Z},\ r \in \mathbb{R} \tag{1}$$

In practice, the following functions are probably the most useful ones:

1) The aforementioned division $f(a,b) = \frac{a}{b}$ with $b \neq 0$. This function is useful in the most cases.
2) Exponential functions, e. g. $f(a,b) = a/1000 \cdot 10^b$. These functions are useful, if numbers from a very big range are to be randomized.

Expressed in SystemVerilog, it leads to a virtual class that features equation 1 as a pure virtual method. We called that method `update()`. Since the code presented below is used in our library A-UVM, we don't call the class `my_class` anymore.

```
virtual class a_uvm_rand_real_base;
  protected real value;
  rand int a;
  rand int b;

  pure virtual function void update();

  function void post_randomize();
    update();
  endfunction

  function real get_value();
    return value;
  endfunction
endclass
```

In order to gain random numbers, an implementation of the function `update()` is to be provided. In the following example, we show a quotient based implementation:

```
class a_uvm_rand_real extends
a_uvm_rand_real_base;
  constraint b_not_zero {b != 0;}

  virtual function void update();
    value = real'(a) / real'(b);
  endfunction
endclass
//...
a_uvm_rand_real m = new();
if(!m.randomize()
  with {a inside {[0:499]}; b == 1000;})
  `uvm_fatal("Rand fail", "...")
```

This solution is already very good, as it allows internal and external constraints in a declarative programming style, where the class is used. Also, in SystemVerilog, object handles can be declared as `rand`. This makes it possible to use the class `a_uvm_rand_real` to be used as a field in any class that needs random real-typed members as if it were a normal `real` field:

```
class data_structure;
  rand a_uvm_rand_real r1, r2;

  constraint c1 {
    r1.a inside {[0:499]}; r1.b == 1000;
  }
  constraint c2 {
    r2.a inside {[0:299]}; r2.b == 1000;
  }

  function new();
    r1 = new();
    r2 = new();
  endfunction
endclass
//...
data_structure d = new();
if(!d.randomize() with {r2.a >= 200;})
  `uvm_fatal("Rand fail", "...")
```

The example shows that the presented SystemVerilog class `a_uvm_rand_real_base` is a powerful and suitable replacement for the missing feature of randomizable real values.

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) [13].

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 [14]. SystemC-AMS allows modeling engineers to describe analog behavior in frequency and time domain.

The drawback is that there is no verification library like UVM for SystemC or SystemC-AMS.

Another new approach is UVM-MS presented in [15] and [16]. 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 overall 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 figure 2) 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 figure 3.

The testbench's driver produces jumps. The height of one jump is the only parameter of the incoming transaction. The height is randomized by the test sequence using the randomization method presented in the previous sections. The monitors calculate the frequency spectra of the model's outputs within an interval of 2ns. 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 calculates the correlation of a 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 leaded to a correlation coefficient greater than 0.89 for every produced frequency spectrum pair. The

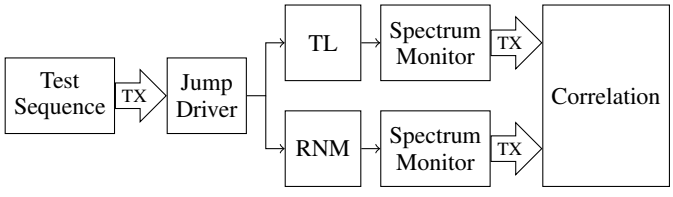

Figure 2. The testbench used in order to compare two different abstraction levels of a voltage regulator

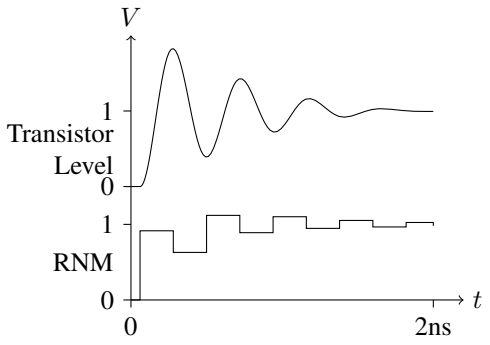

Figure 3. Step responses of the two voltage regulator models. The frequency of the transient oscillation is the same for both models. That indicates, that the frequency spectra of both models can be correlated, in order to evaluate the consistency. If the oscillation frequency of the transistor level circuit was changing due to a change in its design, the correlation would become much smaller, indicating that the RNM is to be updated.

calculation of the correlation coeffcient is shown in [11]. We ran this test sequence in a nightly 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 correlation 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 correlation 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 directed 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 randomizing analog transaction and highlighted its key features. The technique tackles the necessity of being able to stimulate Designs under Verification with analog constrained signals. 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://stacks.iop.org/1748-0221/7/i=01/a=C01093

[10] A. W. Rath, V. Esen, and W. Ecker, "Analog Transaction Level Modeling for Verification of Mixed-Signal-Blocks," *proceedings of DVCON*, March 2012.

[11] A. Rath, V. Esen, and W. Ecker, "Comparison of Analog Transactions using Statistics," in *Proceedings of International Syposium on System-on-Chip*, 2013.

[12] 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

[13] 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

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

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

[16] 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.