Monte Carlo Simulation Using VHDL-AMS

Ekkehart-Peter Wagner
Siemens VDO Automotive AG
Regensburg, Germany

Joachim Haase
Fraunhofer-Institut Integrierte Schaltúngen
Branch Lab EAS Dresden, Germany

Abstract

Monte Carlo simulation is widely used in Spice like circuit simulators. It allows to obtain statistical information derived from estimates of the random variability of circuit parameters. Multiple simulation runs are carried out with different sets of parameters. VHDL-AMS provides flexible possibilities to specify nominal and tolerance values and their distributions. Correlation between parameters can easily be taken into account. This is especially important if behavioral models are considered. The paper describes requirements and implementation aspects of the Monte Carlo simulation using VHDL-AMS.

1 Introduction

Within industrial applications the tolerance- and worst-case-analysis considering all known influencing factors of design parameters are required very often. Reliability and yield of electronic circuits depend on the statistical characteristics of such parameters.

One method for analyzing the effects of tolerances is simulation using Monte Carlo methods. In a Monte Carlo simulation, a mathematical model of a system is repeatedly evaluated. Each run uses different values of system parameters. The selection of the parameter values is made randomly with respect to given distribution functions [1].

Monte Carlo simulation is very time consuming. A lot of simulation runs are required to investigate the behavior of a system subject to the statistical distribution of parameters. Nevertheless, Monte Carlo simulation is very favored simulating electrical circuits and systems. It is widely supported by Spice-like simulation engines. Monte Carlo features are usually available for frequency and time domain analysis [2].

In VHDL-AMS [3] applications it becomes increasingly interesting to make Monte Carlo features available. The basic requirements for statistical simulation linked to VHDL-AMS are summarized by Christen in [4]. His paper concludes that support for statistical modeling can be provided using VHDL packages. He discusses the requirements for Monte Carlo and time series simulation support during the phase of the VHDL-AMS language design. However, at the moment, eight years later, a uniform standard approach to solve these problems in existing VHDL-AMS simulators still does not exist to the knowledge of the authors. Some ideas concerning time series simulation were reported in [5].

In this paper we present *first experiences* how to implement some of the requirements for Monte Carlo simulation [4]

- Usage of the same model for nominal and Monte Carlo analysis
- Assignment of different statistical distributions that are parameterizable to each constant
- Support of continuous and discrete distributions
- Possibility to specify correlation between constants

From a practical point of view the following points should also be mentioned

- Independent random number generation for any constant
- Reproducibility of Monte Carlo simulation within the same simulation tool

Reproducibility of Monte Carlo simulation in different VHDL-AMS simulation tools would be desirable.

We will start with a discussion of the implementation of random number generators for Monte Carlo simulation. Afterwards, we will show how to implement these generators in VHDL-AMS. We will continue with a simple example and conclude with some remarks about further directions.

2 Random Number Generators

2.1 Basic Problems

Initialization of the Pseudo-Random Number Generator

One of the basic problems in Monte Carlo simulation is the generation of random numbers. In Monte Carlo simulations of electrical circuits pseudo-random numbers are typically used. Different approaches to generate such numbers exist. The MATH_REAL package [6] of the IEEE library provides a procedure UNIFORM that returns a pseudo-random number with uniform distribution in the open interval (0, 1). The procedure is declared in the following way:

```
procedure UNIFORM(variable SEED1,SEED2:inout POSITIVE;
                  variable X:out REAL);
```

The algorithm is based on the combination of two multiplicative linear congruential generators. It was published by L'Ecuyer [7]. An advantage of the L'Ecuyer generator is its long period [8]. The VHDL implementation requires the seed values (SEED1, SEED2) to be initialized before the first call to UNIFORM. The seed values are modified after each call to UNIFORM. In order to generate a chain of pseudo-random numbers, the seed values shall be set only in the first call of the procedure (see Annex A.3 of [6]). In the next call the seed values from the previous call have to be used. A different chain of numbers is started every time the seed values are set.

The Ada implementation of the L'Ecuyer generator provides an INITIALIZE procedure that sets two global initial seed values that are updated during every call of the random number generator [8]. An equivalent procedure is not available in the MATH_REAL package. However, a similar functionality is needed in Monte Carlo simulations. Thus, the pseudo-random generator is used to initialize constant objects declared in different design units. The state of the generator has to be passed from one call to the next one by using seed values from a previous call. This can be done in a well-defined way for instance inside a PROCESS statement. The seed values can be held in VARIABLE objects. This approach can not be used e.g. during initialization of generic constants or constants that are declared in different design units.

Thus, another approach has to be used. The state of the random number generator can be held for example in a

- SHARED VARIABLE or a
- FILE.

IEEE DASC P1076a Shared Variable Working Group specified mutually exclusive access semantics for shared variables [9]. If this work could be the base for an extension of the capabilities of random number generation in the IEEE packages. The seeds could be global variables, functions to initialize their values (INIT_SEED) could be provided, and the UNIFORM procedure would have to be modified accordingly. But shared variables are currently not implemented in all available VHDL-AMS simulators.

Due to these existing limitations concerning shared variables we followed the second approach. Seed values are read and written into a file before and after a call of the UNIFORM procedure in the context of Monte Carlo simulation. If at the beginning of a Monte Carlo simulation run the same file is used then the same results will be produced by the simulator. It is assumed that in the elaboration phase (see [3], chapter 12) the calls of the UNIFORM procedure will be carried out in a sequential manner. The elaboration is carried out in the same way prior to the execution phase in every simulation run. Thus, reproducibility is assured if the elaboration phase starts with the same file. On the other hand, every run during Monte Carlo simulation starts with a different file and can work with a different parameter set. The next simulation run starts with the updated file of the last run.

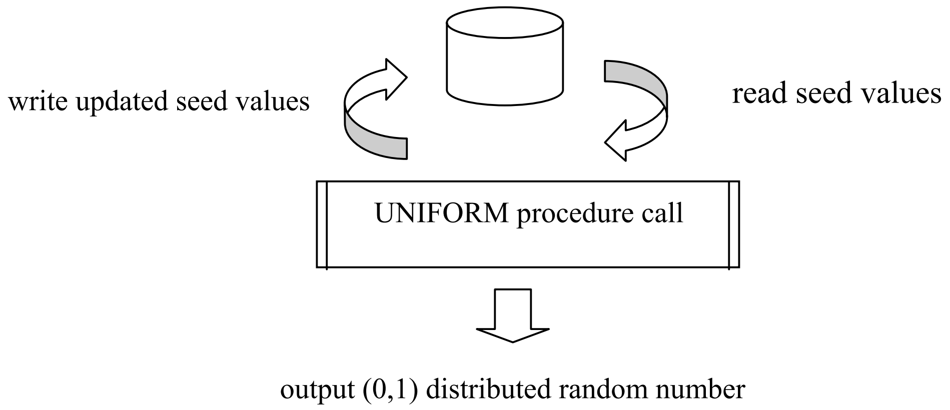

write updated seed values
read seed values
UNIFORM procedure call
output (0,1) distributed random number

Figure 1: UNIFORM procedure call in the elaboration phase of Monte Carlo simulation

This procedure only sufficiently works in the elaboration phase. It is evident that it is not cycle pure. Thus, it can not be applied in the execution phase of a VHDL-AMS simulation.

Transformation of Uniform Random Distribution

Many process and device model parameters are not (0,1) uniform distributed. Generally applied distributions used in Monte Carlo simulation are for example:

- Uniform distribution between a and b (a < b)
- Gaussian distribution N(μ,σ) (also called normal distribution) with mean value μ and standard deviation σ
- Bernoulli distribution having two possible outcomes with probability p=0.5.

Other distributions as triangular and lognormal distributions can also be implemented. Furthermore, the support of user-defined discrete and continuous distributions is expected.

Non-uniform distributed random numbers can be generated using von Neumann's method of generating random samples by evaluating the position of uniform random numbers in a given rectangle or by transformation. The first methods generate samples from any distribution whose probability density function is piecewise continuous and monotonic [10]. It can be used to take user-defined continuous distributions into account. In the second approach, a (0,1) uniform distributed value is transformed through a function to a new value that follows a non-uniform distribution. How this works will be shown in the following examples.

Let X be a (0,1) uniform random distributed number then

$$Y = a + (b - a) \cdot X \tag{1}$$

is a uniform distributed number between a and b.

Let X_1 and X_2 be independent (0,1) uniform distributed numbers then

$$Y_1 = \mu + \sigma \cdot \sqrt{-2 \cdot \ln(X_1)} \cdot \cos(2\pi \cdot X_2) \tag{2}$$

$$Y_2 = \mu + \sigma \cdot \sqrt{-2 \cdot \ln(X_1)} \cdot \sin(2\pi \cdot X_2) \tag{3}$$

are N(μ,σ) normal distributed numbers [11]. Another way is to start with 12 (0,1) uniform distributed numbers X_i (i = 1 … 12) then

$$Y = \mu + \sigma \cdot \left(\sum_{i=1}^{12} X_i - 6 \right) \tag{4}$$

is also N(μ,σ) normal distributed [12].

Let X be a (0,1) uniform distributed number then

$$Y = \begin{cases} v_1 \in \Re & \mathit{for} \quad X \le 0.5 \\ v_2 \in \Re & \mathit{otherwise} \end{cases} \tag{5}$$

is a Bernoulli distributed number with the two real values v_1 and v_2 that occur with the same probability. We can interpret v_1 and v_2 as minimum and maximum of a parameter. We use the name worst case distribution for this distribution in the following.

In the same way, random numbers with other distributions can be generated. Figure 2 shows how to combine these random number generators in VHDL-AMS.

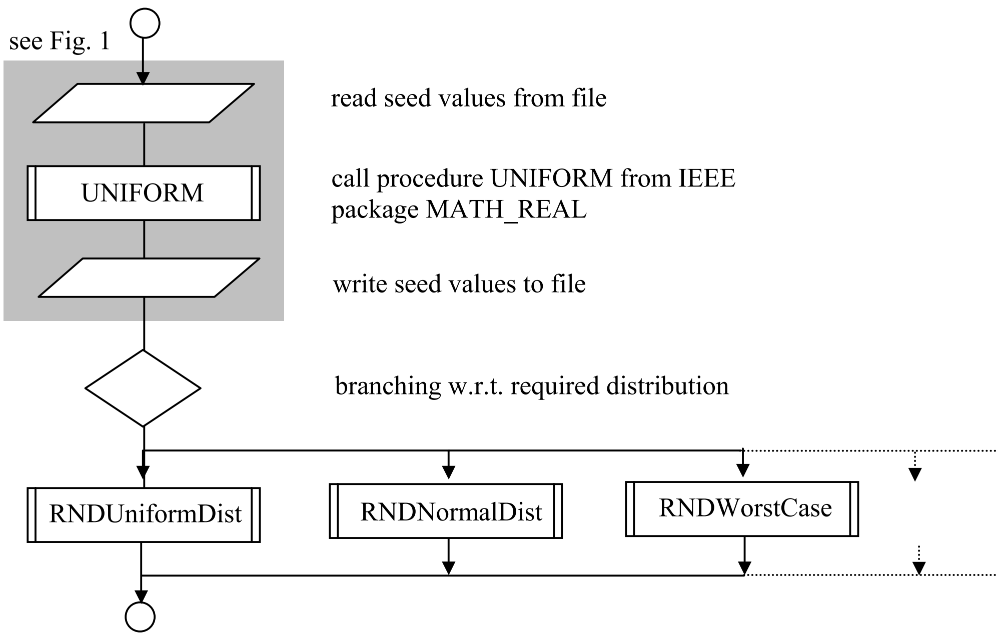

see Fig. 1
read seed values from file
UNIFORM
call procedure UNIFORM from IEEE package MATH_REAL
write seed values to file
branching w.r.t. required distribution
RNDUniformDist
RNDNormalDist
RNDWorstCase

Figure 2: Principle of non-uniform random number generation based on transformation

The mean value of the random numbers is the nominal value of the random constant that has to be initialized during Monte Carlo simulation. Figure 2 describes the main structure of a function RND that can be used to initialize random constants in Monte Carlo simulation runs. The function will be introduced in the section 2.2. Figure 2 can be extended by further general distributions.

Correlation between Random Numbers

In some cases, Statistical circuit simulation requires the consideration of the correlation between random variables. This correlation is described by the correlation matrix *R*. The correlation matrix is a symmetric positive (semi-)definite matrix. Only in the case of Gaussian random numbers it is easy to generate correlated Gaussian random numbers [13].

It is assumed that $Y_1, Y_2, \ldots, Y_n$ shall be Gaussian random numbers with mean values $\mu_1, \mu_2, \ldots, \mu_n$ and standard deviation $\sigma_1, \sigma_2, \ldots \sigma_n$. *R* is the correlation matrix. The element r_{ij} $(-1 \le r_{ij} \le 1)$ describes the correlation between Y_i and Y_j.

To make $Y_1, Y_2, \ldots, Y_n$ available, N(0,1) normal distributed independent random numbers $X_1, X_2, \ldots, X_n$ are generated. A Cholesky decomposition of *R* is carried out, i.e. $R = G^T \cdot G$. Then it follows

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \ldots \\ Y_n \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \ldots \\ \mu_n \end{pmatrix} + \begin{pmatrix} \sigma_1 & 0 & \ldots & 0 \\ 0 & \sigma_2 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & \sigma_n \end{pmatrix} \cdot G^T \cdot \begin{pmatrix} X_1 \\ X_2 \\ \ldots \\ X_n \end{pmatrix} \tag{6}$$

That means in the general case, an algorithm that carries out Cholesky decomposition has to be implemented. In simple cases (i.e. for small *n*) the equation (6) can be solved analytically.

Example

Y_1 and Y_2 are Gaussian random numbers with mean values μ_1 and μ_2, and standard deviation σ_1 and σ_2. The correlation between Y_1 and Y_2 is r_{12}. We get

$$R = \begin{pmatrix} 1 & r_{12} \\ r_{12} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ r_{12} & \sqrt{1-r_{12}^2} \end{pmatrix} \cdot \begin{pmatrix} 1 & r_{12} \\ 0 & \sqrt{1-r_{12}^2} \end{pmatrix} \tag{7}$$

and

$$\begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} + \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ r_{12} & \sqrt{1-r_{12}^2} \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \tag{8}$$

That means

$$Y_1 = \mu_1 + \sigma_1 \cdot X_1 \tag{9.1}$$

$$Y_2 = \mu_2 + \sigma_1 \cdot r_{12} \cdot X_1 + \sigma_2 \cdot \sqrt{1-r_{12}^2} \cdot X_2 \tag{9.2}$$

The equations (9.1) and (9.2) can easily be implemented.

In practice, non-Gaussian data have to be considered in numerous applications. Their probability density function can be expressed in many cases by a truncated Gram-Charlier series expansion using central moments. Different algorithms are proposed to generate correlated non-Gaussian random variables. A special approach that uses the first four central moments is suggested in [14].

The handling of correlated random parameters depends on a lot of requirements that may differ from application to application. VHDL-AMS provides a lot of facilities to support these requirements. However, it seems to be difficult or requires a high effort to make some general methods available to generate correlated non-Gaussian numbers beside trivial cases (e.g. two correlated Gaussian variables). This is, we do not consider these methods in the following.

2.2 Implementation in VHDL-AMS

To realize the functionality described in section 2.1 two VHDL-AMS packages were developed:

- **package** STATISTIC_GLOBAL
- **package** STATISTIC

Both should be compiled into a logical library symbolically named MONTE_CARLO_LIB.

Package STATISTIC_GLOBAL

In the header of the package STATISTIC_GLOBAL two deferred constants are declared

```
constant GLOBAL_STATISTIC     : GLOBAL_STATISTIC_TYPE;
constant GLOBAL_FILE_NAME     : STRING;
```

The first constant allows to decide whether an analysis with nominal values or a Monte Carlo simulation shall be carried out. The enumerated type GLOBAL_STATISTIC_TYPE consists of the values GLOBAL_NOMINAL and GLOBAL_MONTE_CARLO. The initialization of the constant is done in the package body (see also [4]). The constant GLOBAL_FILE_NAME has to be initialized with the relative or full name of the file that carries the seed values (compare Figure 1). The values are saved in ASCII format. Prior to the first simulation, the initial values must meet the requirements concerning SEED1 and SEED2 that are parameters of the UNIFORM procedure [6].

Package STATISTIC

In the package body a function is declared that realizes a random generator with (0,1) distribution:

```
impure function UNIFORM01
  return REAL is
    variable RESULT   : REAL;
    variable SEED     : INTEGER_VECTOR (0 to 1);
  begin
    SEED := READ_SEED;
    UNIFORM (SEED(0), SEED(1), RESULT);
    WRITE_SEED (SEED);
 return RESULT;
 end function UNIFORM01;
```

READ_SEED and WRITE_SEED are two further functions to read and write from a file characterized by the constant GLOBAL_FILE_NAME. The function UNIFORM01 corresponds to Figure 1.

At the moment, the functions RNDUniformDistDIST, RNDNormalDist, and RNDWorstCase that correspond with the equations (1), (4), and (5) resp. are declared. Other distributions will be supplemented in the future.

The code of the function RNDUniformDist demonstrates the implementation of (1). The first parameter is NOMINAL_VALUE that corresponds to the mean value μ. The second parameter TOL determines $a = \mu \cdot (1 - TOL)$ and $b = \mu \cdot (1 + TOL)$ in (1). The third parameter RND01 is transferred from the result of a call of the random number generator UNIFORM01:

```
function RNDUniformDist(NOMINAL_VALUE : REAL; TOL : REAL; RND01 : REAL)
  return REAL is
    variable A      : REAL;
    variable B      : REAL;
    variable RESULT : REAL;
  begin
    A      := NOMINAL_VALUE*(1.0 - TOL);
    B      := NOMINAL_VALUE*(1.0 + TOL);
    RESULT := A + RND01*(B - A);
 return RESULT;
 end function RNDUniformDist;
```

In the header of the package STATISTIC, the functions SET_TOL_UniformDist, SET_TOL_NormalDist, SET_TOL_WorstCase, and RND are made available:

```
-- Set tolerances for uniform distributed values equ. (1)

   function SET_TOL_UniformDist (
       TOL : REAL       -- A = NOMINAL_VALUE*(1.0-TOL)
       )                -- B = NOMINAL_VALUE*(1.0+TOL)
   return TOL_DATA;

-- Set tolerances for normal distributed values equ. (4)

   function SET_TOL_NormalDist (
        SIGMA : REAL   -- standard deviation
        )
   return TOL_DATA;

-- Set tolerances for Bernoulli distribution with p=0.5 equ. (5)

   function SET_TOL_WorstCase (
        TOL : REAL     -- V1 = NOMINAL_VALUE*(1.0-TOL)
        )              -- V2 = NOMINAL_VALUE*(1.0+TOL)
   return TOL_DATA;


-- Function that changes  NOMINAL_VALUE w.r.t. tolerances

   function RND  (
        NOMINAL_VALUE : REAL;
        TOL           : TOL_DATA)
   return REAL;
```

Using the SET_TOL functions a value can be assigned to a data object of the type TOL_DATA that is also declared in the package STATISTIC. By evaluating the data object the type of the distribution and the tolerance values can be determined.

The function RND realizes the flow given by Figure 2. If the constant GLOBAL_STATISTIC from the package STATISTIC_GLOBAL is set to GLOBAL_NOMINAL then the function RND returns the NOMINAL_VALUE. Otherwise, it generates a random number with a mean value that equals the NOMINAL_VALUE and with a distribution given by the second parameter TOL. The TOL parameter can be initialized with the SET_TOL functions.

Usage of the packages

The functions can be used together with existing models. Let us have a look at the VHDL-AMS model of a resistor. P and M are the electrical terminals. The value of the resistance is given by the generic parameter R:

```
library IEEE;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity RESISTOR is
   generic (R : REAL);
   port (terminal P, N : ELECTRICAL);
end entity RESISTOR;

architecture BASIC of RESISTOR is
   quantity V across I through P to N;
begin
   V == R*I;
end architecture BASIC;
```

This model can be instantiated in a VHDL-AMS architecture. The functions that are declared in the header of the package STATISTIC can be used to assign random values to the generic parameter R. This may look like

```
library MONTE_CARLO_LIB;
use MONTE_CARLO_LIB.STATISTIC.all;
…
R1: entity RESISTOR (BASIC)
             generic map (R => RND(5.0E3, SET_TOL_WorstCase(0.01))
             port map    (P => …, N => …)
```

The nominal value of the resistance is 5.0 kΩ. During Monte Carlo Simulation 5.0 kΩ +/– 1% are used. Following this approach, existing models can be used in Monte Carlo simulation. Furthermore, it is also possible to define special architectures that describe elements with given tolerances. For instance, the following architecture TEN_PERC describes a resistor with 10 % tolerance:

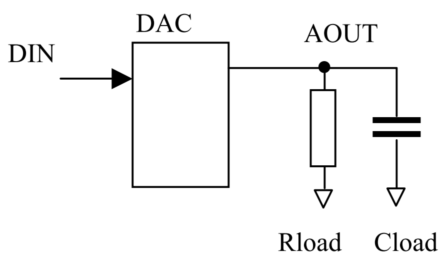
```
library MONTE_CARLO_LIB;
use MONTE_CARLO_LIB.STATISTIC.all;
architecture TEN_PERC of RESISTOR is
   constant TOL : TOL_TYPE := SET_TOL_WorstCase(0.1);
   constant RES : REAL     := RND(R, TOL);
   quantity V across I through P to N;
begin
   V == RES*I;
end architecture TEN_PERC;
```

This model can then be instantiated without special knowledge of the statistical packages:

```
…
R1: entity RESISTOR (TEN_PERC)
             generic map (R => 5.0E3)
             port map    (P => …, N => …)
...
```

3 Examples

One of the advantages of using Monte Carlo simulation with VHDL-AMS is the possibility to apply it on mixed-signal circuits. Figure 3 shows a typical example.



VHDL-AMS model (extract)

```
-- 1 % tolerance
constant t1 : TOL_DATA
   := SET_TOL_WorstCase(0.01) ;
…
constant Rload : REAL
   := RND (1.0E6, t1);
constant Cload : REAL
   := RND (1.0E-12, t1);
```

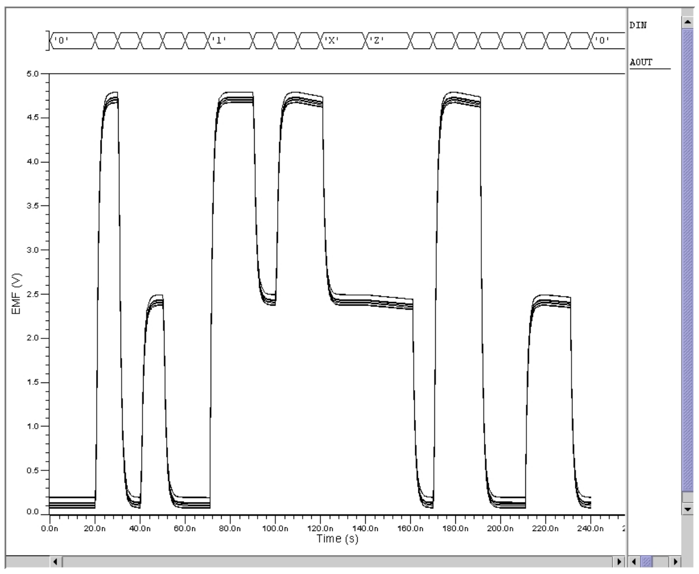
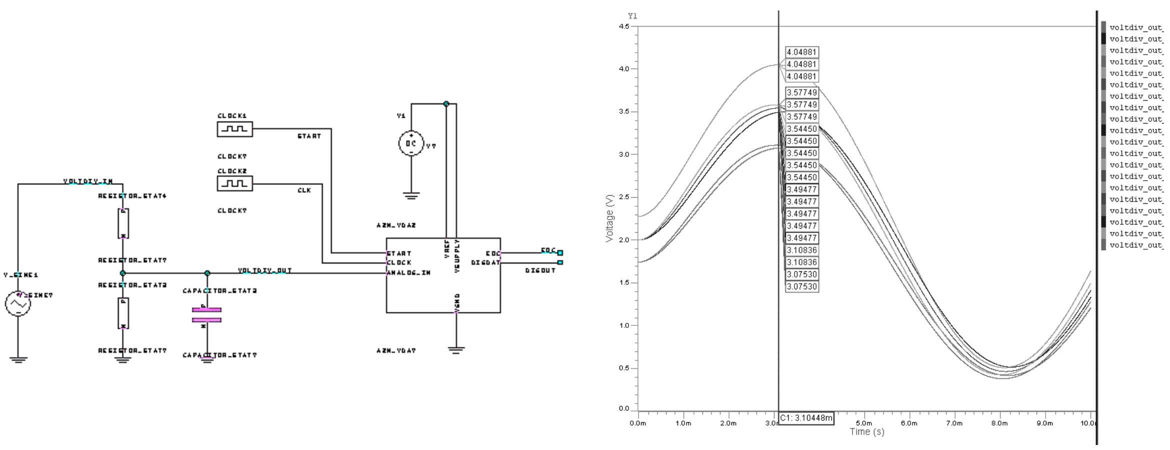


Figure 3: DAC with input signal DIN and voltages at AOUT

Values of load resistor and load capacitor are random parameters. The statistical influence of output resistors of the DAC is investigated in the Monte Carlo simulation.



. Figure 4: ADC channel and voltages at analog input

The circuit represented in Fig. 4 was investigated for a discrete distribution of the lumped R and C elements and Gaussian distributed leakage current of the AD converter.

The examples were carried out with the multi-run features of the programs ADVance MS and SystemVision of Mentor Graphics [15].

4 Further Directions

We gained first experiences with Monte Carlo simulation using VHDL-AMS. Further work will include other probability distributions. We will also include user-defined discrete and piecewise-linear distributions. We also have to check the quality of the generated numbers.

To our opinion, it should be checked whether the definition of statistical packages for Monte Carlo simulation could be part of further activities of the 1076.1 working group. A problem to be solved in a unified and easy way particularly concerns the initialization of UNIFORM or an equivalent procedure and the update of the seed values in the Monte Carlo simulation runs. It should be assured that Monte Carlo simulation using VHDL-AMS delivers the same results in different simulators.

Simulators should support Monte Carlo simulation of VHDL-AMS descriptions. Some aspects are for instance

- Supplement of multi-run-simulations into the list of available analyses. The simulation program should know that the Monte Carlo feature is used. This could avoid unnecessary repetition of some of the stages of the evaluation phase as for instance reading the netlist.
- Support of the initialization of (global) seed values in an easy way
- Implementation of statistical post-processing-tools (for generating histograms, calculating envelopes, mean values, variances, ...)

Furthermore, the usage of Monte Carlo simulation together with the behavioral modeling language VHDL-AMS opens a lot of other opportunities. For instance, results from Monte Carlo simulation could be used for the generation of response surface models [16]. In this case, parameters and selected simulation results of each run should be saved and evaluated afterwards. One could also influence the generation of parameters for different simulation runs by some add-on tools. There is no limit to other ideas.

References

[1] O'Connor P.D.T.: Practical Reliability Engineering. Chichester: John Wiley & Sons Ldt, 2002.

[2] Vlach, J.; Singhal, K.: Computer Methods for Circuit Analysis and Design. New York: Van Nostrand Reinhold, 1994.

[3] IEEE Standard VHDL Analog and Mixed-Signal Extensions (IEEE Std 1076.1-1999). Approved 18 March 1999. Available: http://www.designers-guide.com/Modeling/1076.1-1999.pdf

[4] Christen, E.: Statistical Modeling.
Available: http://www.vhdl.org/analog/wwwpages/language_proposal/STAT.html

[5] Monnerie, G.; Lewis, N.; Dallet, D.; Levi, H. ; Robbe, M. : Modelling of transient noise sources with VHDL-AMS and normative spectral interpretation. Proc. Forum on Specification & Design Languages FDL'03, September 23-26, 2003, Frankfurt/M., pp. 108-119.

[6] IEEE Standard VHDL Mathematical Packages (IEEE Std 1076.2-1996). Approved 19 September 1996.

[7] L'Ecuyer, P.: Efficient and Portable Combined Random Number Generators. Communications of the ACM 31(1988)6, pp. 742-774.

[8] Graham, W.N.: A Comparison of Four Pseudo Random Number Generators Implemented in Ada. ACM SIGSIM Simulation Digest 22(1992)2, pp. 3-18.

[9] http://www.eda.org/svwg/

[10] Forsythe, G.E.: Von Neumann's comparison method for random sampling from the normal and other distributions. Report CS-TR-72-254. Stanford University, 1972.
Available: ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/72/254/CS-TR-72-254.pdf

[11] Box, G.E.P.; Muller, M.E.: A Note on the Generation of Random Normal Deviates.Annals Math. Stat. 29(1958), pp. 610-611.

[12] Schrüfer, E.: Signalverarbeitung. München-Wien: Carl Hanser Verlag, 1990.

[13] Esbaugh, K.S.: Generation of correlated parameters for statistical circuit simulation. Trans. on CAD $\underline{11}$(1992)10, pp. 1198-1206.

[14] Karvanen, J. : Generation of Correlated Non-Gaussian Random Variables from Independent Components. Proc. 4th Int. Symposium on Independent Component Analysis and Blind Signal Separation ICA 2003, April 2003, Nara (Japan), pp. 769-774.

[15] http://www.mentor.com/system

[16] Box, G.E.P.; Draper, N.R.: Empirical Model-Building and Response Surfaces. New York: John Wiley & Sons, 1987.