Appendix A
Standard Formats for Circuit Characterization

A.1 Standard Parasitic Exchange Format SPEF

Standard Parasitic Exchange Format (SPEF) is an ASCII format that represents data of wires in a chip. The latest definition of the format is given by the IEEE Std 1481–2009 [1]. It defines syntactical forms to describe resistances (*RES), capacitances (*CAP), and inductances (*INDUC) of wires that are considered as parasitics. SPEF information can also capture dependency of parasitics on technology parameters (*VARIATION_PARAMETERS). The data can be determined by layout parasitic extraction or package/board extraction tools. These tools evaluate the floorplanning information. They can consider sheet resistances of each interconnect layers and contact resistances of each via cell. Usually, capacitance tables are used to determine the area capacitances of nets to substrate, the fringe or side-wall capacitances to substrate and the cross-coupling capacitances between nets.

Based on SPEF data, parasitic information can be exchanged between different tools. A general SPEF description is structured into

- Header definition
- Name map definition
- Power and ground net definition
- External definition
- Hierarchical SPEF (entities) definition
- Process and temperature variation definition
- Internal definition

Header and internal definitions are mandatory.

The header definition describes administrative issues as used SPEF version (*SPEF), design name (*DESIGN) and others. It contains information about used lexical elements (*DIVIDER, *DELIMITER) and scaling factors and units for time (*T_UNIT) and capacitances (*C_UNIT), resistances (*R_UNIT), and inductance (*L_UNIT). The name map (*NAME_MAP) allows to map a positive integer number to a name that may be used multiple times in an SPEF file. The power

M. Dietrich and J. Haase (eds.), *Process Variations and Probabilistic Integrated Circuit Design*, DOI 10.1007/978-1-4419-6621-6,

and the ground net definitions (*POWER_NETS, *GROUND_NETS) may specify references to power and ground nets.

The hierarchical SPEF definition (*DEFINE, *PDEFINE) can be used to reference entity instances within the current SPEF file. This allows merging multiple SPEF files.

The external definition defines external logical ports (*PORTS) and physical-only ports (*PHYSICAL_PORTS) of the nets that are described.

The internal definition of the nets that represent the wires can be done in detailed (*D_NET, *D_PNET) or reduced form (*R_NET, *R_PNET). The connection section (*CONN) of a detailed net describes the external (*P) and internal (*I) connection points of a net and capacitances, resistances, and inductances that built up the net. The reduced form is based on a pi-model that consists of two capacitances and a resistance (*C2_R1_C1). Distributed models are reduced to equivalent reduced models by asymptotic waveform evaluation (AWE). The reduced models represent an admittance model seen by the driving cell. The description of a reduced net requires information on the driver (*DRIVER, *CELL).

The SPEF data can be used for circuit simulation, power calculation, and crosstalk analysis for instance. In digital circuit verification, delays are determined based on these data. SPEF information can be used to do postlayout Static Timing Analysis. Rule checks and power calculation are other areas of application.

Example:

```
// Header definition

*SPEF         "IEEE 1481-2009"
*DESIGN       "Sample"
*DATE         "Monday December 18, 1995"
*VENDOR       "Sample Tool"
*PROGRAM      "Sample Generator"
*VERSION      "1.1.0"
*DESIGN_FLOW "Sample Flow"
*DIVIDER /
*DELIMITER :
*BUS_DELIMITER [ ]
*T_UNIT 1 NS
*C_UNIT 1 PF
*R_UNIT 1 OHM
*L_UNIT 1 HENRY

// Name map definition

*NAME_MAP

*1  in1
```

```
*2  out1
*3  net1
*4  net2
*5  net3
*6  net4


// External ports

*PORTS

*1 I
*2 O

// Detailed of one net description

*D_NET  *4  0.287695

*CONN
*I *5:Z  O  *D DRIVER_CELL_1
*I *6:A  I  *D DRIVER_CELL_1

*CAP
1 *5:Z      0.189802
2 *6:A      0.097893

*RES
1 *5:Z *6:A 1.054678
*END

...
```

Listing A.1 Part of a SPEF file based on [1]

The detailed net net2 (identified by *4) connects point Z of net3 with point A of net4. The parameters are determined for a connection with DRIVER_CELL_1. The letters I and O in the definition of ports and connection points of nets characterize inputs and outputs.

The variation definition (*VARIATION_PARAMETERS) is new in the IEEE Std 1481–2009 compared to older versions of the standard. It defines the process parameters that affect capacitances, inductances, and resistances of interconnects. Furthermore, first-order and second-order coefficients (CRT1, CRT2) for the temperature dependencies of resistances are defined as well as the nominal temperature for the extraction. The variation effects are described by

$$C\left(\underline{p}\right) = C_0 \cdot \frac{1+\sum_j cn_j \cdot v_j}{1+\sum_i cd_i \cdot v_i} \tag{A.1}$$

$$L\left(\underline{p}\right) = L_0 \cdot \frac{1+\sum_j ln_j \cdot v_j}{1+\sum_i ld_i \cdot v_i} \tag{A.2}$$

$$R\left(\underline{p},T\right) = R_0 \cdot \left(1+a\cdot(T-T_0)+b\cdot(T-T_0)^2\right) \cdot \frac{1+\sum_j rn_j \cdot v_j}{1+\sum_i rd_i \cdot v_i}. \tag{A.3}$$

$\underline{p}$ is a vector of process parameters. p_i is a component of this vector. T_0 is the nominal temperature for the extraction of the parameters. T is the current temperature. C_0, L_0, T_0 are capacitance, inductance, and resistance values at nominal values of the process parameters and nominal temperature.

$cn_j = \frac{1}{C_0} \cdot \frac{\partial C(\underline{p})}{\partial p_j} \cdot NF(p_j)$, $ln_j = \frac{1}{L_0} \cdot \frac{\partial L(\underline{p})}{\partial p_j} \cdot NF(p_j)$, $rn_j = \frac{1}{R_0} \cdot \frac{\partial R(\underline{p})}{\partial p_j} \cdot NF(p_j)$ are sensitivity coefficients for so-called N-type variation parameters. These coefficients characterize the numerators in the expressions that describe the parameter dependencies of capacitance, inductance, and resistance. $NF(p_j)$ is an optional normalization factor of the process parameter p_j.

$cd_i = C_0 \cdot \frac{\partial C^{-1}(\underline{p})}{\partial p_i} \cdot NF(p_i)$, $ld_i = L_0 \cdot \frac{\partial L^{-1}(\underline{p})}{\partial p_i} \cdot NF(p_i)$, $rd_i = R_0 \cdot \frac{\partial R^{-1}(\underline{p})}{\partial p_i} \cdot NF(p_i)$ are sensitivity coefficients for so-called D-type variation parameters. These coefficients characterize the denumerators in the expressions that describe the parameter dependencies of capacitance, inductance, and resistance. A process parameter is either a N type or an D-type parameter for the description of capacitances, inductances, and resistances resp. It is also possible that a variation of a process parameter does not influence neither numerator nor denumerator. Then it is called X-type variation parameter.

$a = \frac{1}{R_0} \cdot \frac{\partial R}{\partial T}$ and $b = \frac{1}{R_0} \cdot \frac{\partial^2 R}{\partial T^2}$ are sensitivity coefficients that characterize the temperature dependency of resistances (CRT1, CRT2).

The worst case variation $\Delta v_i = VC(p_i) \cdot VM(p_i)$ of a process parameter is given by its variation coefficient $VC(p_i)$ and the variation multiplier $VM(p_i)$. Nominal values of the variation parameters p_i are assumed to equal the mean values $\mu(p_i)$ of the associated probability distribution. The relation $VC(p_i) = \frac{\sigma(p_i)}{NF(p_i)}$ gives the standard deviation σ of the distribution.

Example:

The subsequent example is taken from [1]. The variation definition is given by

```
*VARIATION_PARAMETERS
0 "field_oxide_T"  D X X   0.080 1
1 "poly_T"         D X X   0.030 1
2 "poly_W"         D X X   0.023 1
3 "Diel1_T"        X X D   0.050 1
4 "metal1_T"       X N X   0.050 1
5 "metal1_W"       X N X   0.030 1
6 CRT1
```

```
7 CRT2
27.0000
```

Listing A.2 Variation parameters definition in a SPEF file [1]

The first lines characterize process parameters as thickness, width, and permittivity of dielectric layers by its parameter index i, a string and the variation parameters types (N, D, or X) for capacitance, resistance, and inductance followed by the variation coefficient ($VC(p_i)$) and the normalization factor ($NF(p_i)$). Afterward, the parameter indices for first- and second-order temperature sensitivities of resistances are given. The last value is the nominal temperature.

The characterization of capacitances, inductances, and resistances can now be extended by its sensitivities. The variation description follows after *SC. It is given by pairs of parameter index and then associated sensitivity coefficient.

```
*CAP
1 *5:Z       0.189802  *SC 0:-0.005 1:0.029 2:0.026
...

*RES
1 *5:Z *6:A 1.054678  *SC 4:0.900  5:0.531 6:0.00321
7:-0.00021
```

Listing A.3 Sensitivity description in an SPEF file based on [1]

A.2 Standard Delay Format (SDF)

The Standard Delay Format (SDF) is a standardized format to describe delay and timing information of a digital design [2]. It describes path delays, timing constraint values, interconnect delays, and high level technology parameters in a tool independent way. It is commonly used to calculate data from postlayout information. However, it can also provide constraints for a prelayout design step. One of the original intentions of this development was to provide delay information for the digital simulation, for instance, using Verilog.

The SDF file is an ASCII file. It starts with a header section where some administrative information as SDFVERSION, DESIGN name, DATE, VENDOR and the name of the PROGRAM name that created the file are given. PROCESS, VOLTAGE, and TEMPERATURE in degrees Celsius are specified. All these values do not affect the meaning of the data. Thus, this part of the header section is often used to characterize the Process-Voltage-Temperature (PVT) conditions. They are provided for documentation purposes The TIMESCALE defines the unit of timing information.

IOPATH delays characterize the input–output path delays of a device (CELL). An IOPATH is defined by an input (or bidirectional) port and an output (or bidirectional) port of a device followed by a list of delays. The delay values are given in general by triples characterizing the minimum, typical, and maximum value of a delay.

The list of delays contains either one, two, three, six, or twelve elements. Each token corresponds to a special output transition (see [2, Table 1]). The delay is applied when the output port (for IOPATH or INTERCONNECT) or in the case of two elements, the first element describes the $0 \rightarrow 1$ transition. The second element characterizes the $1 \rightarrow 0$ transition. The delay is considered at the end of the path. Furthermore, it is possible to specify delays regarding special conditions (COND) at the input port of a device. On the opposite side, all delays of a device can be described by only one list that starts with the keyword DEVICE. Delays can be given by ABSOLUTE or INCREMENT values. Incremental delays are added to existing values in the same SDF file whereas absolute values replace old values. This may be used to characterize several instances (INSTANCE) of the same CELLTYPE by describing the differences between the instances.

Interconnect delays characterize the wires between outputs and inputs of devices. They can be given as PORT delays, NETDELAYs, or INTERCONNECT delays. Port delays are considered as delays at input (or bidirectional) ports of a device. Net delays specify the delays from all drivers to all loads of a net in the same way. A more detailed description is given by INTERCONNECT delays. In this case, the first port is a driver port (output or bidirectional port) and the second port is an input (or bidirectional) port of a driven device.

Furthermore, SDF files can contain statements considering timing constraints of pulses as setup and hold times, pulse width, period and others.

Example:

```
(DELAYFILE
   (SDFVERSION "4.0")
   (DESIGN     "Test")
   (DATE       "Tue Sep 16 15:45:26 2008")
   (VENDOR     "Test vendor")
   (PROGRAM    "Test Tool")
   (VERSION     "1.0")
   (DIVIDER /)
   (VOLTAGE     1.00::1.00)
   (PROCESS     "typical")
   (TEMPERATURE 25.00::25.00)
   (TIMESCALE 1ns)
   (CELL
     (CELLTYPE "SYSTEM")
     (INSTANCE)
     (DELAY
        (ABSOLUTE
           (INTERCONNECT a0    c1/a  (0.004::0.005)
             (0.004::0.005))
           (INTERCONNECT c1/z  c2/a  (0.002::0.003)
             (0.002::0.003))
```

```
            (INTERCONNECT c2/z  b0    (0.001::0.001)
              (0.001::0.001))
          )
       )
     )

    (CELL
      (CELLTYPE "INV")
      (INSTANCE c1)
      (DELAY
         (ABSOLUTE
             (IOPATH A Z (0.143::0.167) (0.032::0.152))
          )
       )
     )

    (CELL
      (CELLTYPE "INV")
      (INSTANCE c2)
      (DELAY
         (ABSOLUTE
             (IOPATH A Z (0.129::0.157) (0.122::0.142))
          )
       )
     )

)
```

Listing A.4 Principal structure of an SDF file

The SDF information can be used to specify delay times of digital models used in VHDL or Verilog simulations. It can also be used to investigate critical paths applying STA tools.

The current version of the file format [2] does not support dependencies of delays on process parameters as well as dependencies on supply voltages of a cell. It only considers worst-case conditions. Using minimal and maximal delay times, the estimation of delays of interesting paths of a design may be either too optimistic or too pessimistic if inter-die variations have to be taken into account. The "engineering approach" to solve this problem is to provide derating tables that correct worst-case values. This approach is known as On-Chip Variation (OCV) analysis. The improved version is called Advanced On-Chip Variation (AOCV) that uses variable derating factors based on the digital levels and location of a cell [3].

A.3 Nonlinear Delay Model NLDM

The nonlinear delay model is widely used to characterize the propagation delay through digital cells and blocks to their outputs depending on the load capacitances and input rise and fall times. It also describes how output rise and fall times depend on these values (see also Sect. 4.1). NLDM was introduced in the early 1990s [4]. The model characterizes the propagation delay through a cell and the rise and fall times at the output ports.

The general model description is part of the Liberty format specification that is an industry's used library modeling standard [5, 6]. Members of the Liberty Technical Advisory Board (LTAB) are EDA vendors such as Magma Design Automation, Mentor Graphics and Synopsys and semiconductor companies such as AMD, IBM, Infineon and Texas Instruments.

The Liberty description starts with library-level attributes as, for instance, the `technology`, the `delay_model`, the units of pulling resistances, currents, time, voltages, and capacitances. The nonlinear delay model is a `table_lookup` model. Information is given concerning the operating conditions. Different library files can be created for best and worst cases. Default values for pin attributes as `default_inout_pin_cap` are provided. The `wire_load` group information may be used to estimate interconnect parameters depending on the fanout of a cell.

Furthermore, the threshold levels of the input (`input_threshold_pct_rise`, `input_threshold_pct_fall`) and output signals (`output_threshold_pct_rise`, `output_threshold_pct_fall`) are given and are used to determine delays provided by the library. The level of the threshold points that are used to determine rise and fall times is also given by `slew_lower_threshold_pct_rise` and `slew_upper_threshold_pct_rise` for a rising edge and `slew_upper_threshold_pct_fall` and `slew_lower_threshold_pct_fall`, resp.

The cell descriptions characterize electrical properties of input and output pins as well as timing conditions. The `rise_capacitance` and the `fall_capacitance` attributes allow to specify different capacitance values for input or inout pins for rising and falling waveforms, resp. The capacitances for output pins are not specified if their effects are considered in the timing information of a cell. The electrical configuration at the output (for instance, pull-up or pull-down resistances) can be described by the `driver_type` attribute. The nominal relations of the NLDM model are given by tables that characterize the delays in the case of rising and falling edges at outputs (`cell_rise`, `cell_fall`. `rise_transition` and `fall_transition` tables describe the dependencies of rising and falling times at output pins on input slopes and load capacitances. The tables allow to describe nonlinear relations by selected points.

New attributes can be created using the `define` statement. This property can be used to characterize first-order sensitivities of dependent table values on process parameters as for instance `cell_rise_sensitivity` (see also Sect. 4.5.2.2). This

offers the opportunity to consider parameter variations in the analysis. However, the accuracy is limited if only first-order sensitivities are used.

The delay information can be extended by a Nonlinear Power Model (NLPM) that was introduced in 1995 [7]. It describes the leakage power per state (`cell_leakage_power`). The power that is dissipated within a cell if the signal of an output port changes is considered as `internal_power`. Dependencies on process parameters can be taken into account by first-order sensitivities for instance (see also Sect. 4.5.2).

Examples of libraries are provided in [8].

Example:

```
library (MyCellLibrary) {

  /* general */
  technology(cmos);
  delay_model              : table_lookup;
  in_place_swap_mode       : match_footprint;
  library_features(report_delay_calculation,report_power_calculation);

  /* units */
  time_unit                : "1ns";
  leakage_power_unit       : "1nW";
  voltage_unit             : "1V";
  current_unit             : "1uA";
  pulling_resistance_unit  : "1kohm";
  capacitive_load_unit      (1,pf);
  /* internal power unit is 1pJ */

  /* operating conditions */
  nom_process              : 1.0;
  nom_temperature          : 25.0;
  nom_voltage              : 1.1;

  operating_conditions (typical) {
    process                : 1.00;
    voltage                : 1.10;
    temperature            : 25.00;
    tree_type              : balanced_tree;
  }
  default_operating_conditions : typical;

  /* thresholds */
  slew_lower_threshold_pct_fall : 30.0;
  slew_lower_threshold_pct_rise : 30.0;
  slew_upper_threshold_pct_fall : 70.0;
  slew_upper_threshold_pct_rise : 70.0;
  slew_derate_from_library      : 1.0;
  input_threshold_pct_fall      : 50.0;
  input_threshold_pct_rise      : 50.0;
  output_threshold_pct_fall     : 50.0;
  output_threshold_pct_rise     : 50.0;

  /* leakage */
  default_leakage_power_density : 0.0;
  default_cell_leakage_power    : 0.0;

  /* pin capacitances */
  default_inout_pin_cap       : 1.0;
  default_input_pin_cap       : 1.0;
  default_output_pin_cap      : 0.0;
  default_fanout_load         : 1.0;

  /* wire loads */
  default_wire_load_capacitance : 0.000177;
  default_wire_load_resistance  : 0.0036;

  power_lut_template (slp_load_pwr) {
    variable_1 : input_transition_time;
    variable_2 : total_output_net_capacitance;
    index_1 ("0.0060, 0.0200, 0.0400, 0.0800, 0.1400, 0.2800, 0.5600")
    index_2 ("0.0004, 0.0008, 0.0016, 0.0032, 0.0064, 0.0128, 0.0256")
  }

  lu_table_template (slp_load_tmg) {
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    index_1 ("0.0060, 0.0200, 0.0400, 0.0800, 0.1400, 0.2800, 0.5600")
```

```
    index_2 ("0.0004, 0.0008, 0.0016, 0.0032, 0.0064, 0.0128, 0.0256")
  }


  /*****************************************************************
    BEGIN Additional Definitions for Process Parameters and Sensitivities   */

  power_lut_template (slp_load_pwr_sensitivity) {
    variable_1 : input_transition_time;
    variable_2 : total_output_net_capacitance;
    index_1 ("0.0060, 0.0200, 0.0400, 0.0800, 0.1400, 0.2800, 0.5600")
    index_2 ("0.0004, 0.0008, 0.0016, 0.0032, 0.0064, 0.0128, 0.0256")
  }

  lu_table_template (slp_load_tmg_sensitivity) {
    variable_1 : input_net_transition;
    variable_2 : total_output_net_capacitance;
    index_1 ("0.0060, 0.0200, 0.0400, 0.0800, 0.1400, 0.2800, 0.5600")
    index_2 ("0.0004, 0.0008, 0.0016, 0.0032, 0.0064, 0.0128, 0.0256")
  }

  define_group(cell_fall_sensitivity,       timing);
  define_group(cell_rise_sensitivity,       timing);
  define_group(fall_transition_sensitivity, timing);
  define_group(rise_transition_sensitivity, timing);
  define_group(fall_power_sensitivity,      internal_power);
  define_group(rise_power_sensitivity,      internal_power);

  define(param_name, cell_fall_sensitivity, string);
  define(values,     cell_fall_sensitivity, string);
  define(param_name, cell_rise_sensitivity, string);
  define(values,     cell_rise_sensitivity, string);
  define(param_name, fall_transition_sensitivity, string);
  define(values,     fall_transition_sensitivity, string);
  define(param_name, rise_transition_sensitivity, string);
  define(values,     rise_transition_sensitivity, string);
  define(param_name, fall_power_sensitivity, string);
  define(values,     fall_power_sensitivity, string);
  define(param_name, rise_power_sensitivity, string);
  define(values,     rise_power_sensitivity, string);

  define_group(process_parameter, library);
  define(parameter_type,     process_parameter, string);
  define(distribution_type,  process_parameter, string);
  define(nominal_value,      process_parameter, float);
  define(sigma,              process_parameter, float);

  /* process parameters used */
  process_parameter (NMOS_THKOX) {
    parameter_type    : inter_cell;
    distribution_type : normal;
    nominal_value     : 0.0;
    sigma             : 1.0;
  }

  process_parameter (NMOS_VTH) {
    parameter_type    : inter_cell;
    distribution_type : normal;
    nominal_value     : 0.0;
    sigma             : 1.0;
  }

  /* END   Additional Definitions for Process Parameters and Sensitivities
   *****************************************************************************/


cell (INV1) {
  cell_leakage_power : 2.75;

  leakage_power () {
    when        : "!A";
    value       : 2.0;
  }
  leakage_power () {
    when        : "A";
    value       : 3.5;
  }

  pin (A) {
    direction        : input;
    capacitance      : 0.00046;
    fall_capacitance : 0.00045;
    rise_capacitance : 0.00048;
    fall_capacitance_range(0.00042, 0.00054);
    rise_capacitance_range(0.00043, 0.00057);
    max_transition   : 0.56;
  }

  pin (Z) {
```

```
direction        : output;
max_capacitance  : 0.0256;
min_capacitance  : 0.0;
function         : "(!A)";

timing () {
  related_pin  : "A";
  timing_sense : negative_unate;

  cell_fall(slp_load_tmg) {
    values ("0.01041, 0.01289, 0.01753, 0.02698, 0.04588, 0.08364, 0.15914", \
            "0.01393, 0.01721, 0.02261, 0.03201, 0.05085, 0.08869, 0.16405", \
            "0.01637, 0.02087, 0.02837, 0.04026, 0.05933, 0.09693, 0.17235", \
            "0.01755, 0.02363, 0.03386, 0.05022, 0.07531, 0.11409, 0.18905", \
            "0.01514, 0.02325, 0.03705, 0.05922, 0.09361, 0.14526, 0.22330", \
            "0.00347, 0.01465, 0.03324, 0.06311, 0.10955, 0.18021, 0.28516", \
            "-0.02680, -0.01241, 0.01189, 0.05177, 0.11436, 0.21013, 0.35336");
  }
  cell_rise(slp_load_tmg) {
    values ("0.01745, 0.02223, 0.03174, 0.05069, 0.08852, 0.16404, 0.31499", \
            "0.02353, 0.02814, 0.03748, 0.05631, 0.09405, 0.16957, 0.32059", \
            "0.03144, 0.03757, 0.04776, 0.06620, 0.10366, 0.17900, 0.32979", \
            "0.04235, 0.05055, 0.06446, 0.08673, 0.12380, 0.19840, 0.34891", \
            "0.05863, 0.06915, 0.08730, 0.11734, 0.16420, 0.23887, 0.38794", \
            "0.08429, 0.09789, 0.12103, 0.15989, 0.22266, 0.31893, 0.46917", \
            "0.12912, 0.14531, 0.17433, 0.22362, 0.30471, 0.43312, 0.62828");
  }
  fall_transition(slp_load_tmg) {
    values ("0.00526, 0.00721, 0.01131, 0.01951, 0.03587, 0.06863, 0.13408", \
            "0.00815, 0.00989, 0.01242, 0.01955, 0.03590, 0.06864, 0.13404", \
            "0.01219, 0.01424, 0.01767, 0.02282, 0.03620, 0.06870, 0.13418", \
            "0.01903, 0.02161, 0.02578, 0.03285, 0.04392, 0.06965, 0.13433", \
            "0.03093, 0.03414, 0.03944, 0.04859, 0.06346, 0.08612, 0.13645", \
            "0.05212, 0.05624, 0.06353, 0.07526, 0.09402, 0.12459, 0.17059", \
            "0.09276, 0.09723, 0.10614, 0.12155, 0.14593, 0.18511, 0.24687");
  }
  rise_transition(slp_load_tmg) {
    values ("0.01144, 0.01602, 0.02496, 0.04296, 0.07869, 0.15051, 0.29392", \
            "0.01269, 0.01636, 0.02502, 0.04293, 0.07873, 0.15055, 0.29330", \
            "0.01752, 0.02062, 0.02692, 0.04298, 0.07884, 0.15062, 0.29354", \
            "0.02490, 0.02926, 0.03639, 0.04850, 0.07920, 0.15058, 0.29353", \
            "0.03628, 0.04177, 0.05146, 0.06731, 0.09168, 0.15170, 0.29424", \
            "0.05622, 0.06231, 0.07444, 0.09537, 0.12874, 0.17894, 0.29695", \
            "0.09468, 0.10007, 0.11377, 0.13889, 0.18245, 0.25140, 0.35281");
  }
  /* Additional Sensitivity Tables for Timing  */
  cell_fall_sensitivity (slp_load_tmg_sensitivity) {
    param_name : NMOS_VTH;
    values : " 0.00042152, 0.00061292, 0.00082434, 0.00123816, 0.00195448, 0.00344520, 0.00667040, \
               0.00081422, 0.00095084, 0.00108636, 0.00144650, 0.00214434, 0.00355960, 0.00667260, \
               0.00194458, 0.00219032, 0.00246400, 0.00277860, 0.00332640, 0.00454740, 0.00736340, \
               0.00353100, 0.00379060, 0.00386320, 0.00464420, 0.00524260, 0.00627000, 0.00869220, \
               0.00646360, 0.00691240, 0.00724460, 0.00771100, 0.00869880, 0.01028940, 0.01232440, \
               0.00911680, 0.00953040, 0.01027400, 0.01072060, 0.01156980, 0.01416360, 0.01634160, \
               0.01423840, 0.01553860, 0.01612600, 0.01684980, 0.01824020, 0.01956240, 0.02402400";
  }
  cell_rise_sensitivity (slp_load_tmg_sensitivity) {
    param_name : NMOS_VTH;
    values : " 0.00003597, 0.00003012, 0.00002226, 0.00001047,-0.00000152,-0.00001102,-0.00001727, \
              -0.00000290, 0.00001552, 0.00002130, 0.00002171, 0.00001459, 0.00000237,-0.00000820, \
              -0.00047982,-0.00030338,-0.00020211,-0.00011458,-0.00005896,-0.00002807,-0.00001761, \
              -0.00160600,-0.00111320,-0.00086636,-0.00059796,-0.00033418,-0.00017963,-0.00010384, \
              -0.00372900,-0.00317240,-0.00272140,-0.00203962,-0.00137918,-0.00078716,-0.00043274, \
              -0.00603460,-0.00533500,-0.00481580,-0.00389620,-0.00286220,-0.00174218,-0.00096470, \
              -0.01102860,-0.00999460,-0.00937640,-0.00815320,-0.00630300,-0.00442860,-0.00259380";
  }
  fall_transition_sensitivity (slp_load_tmg_sensitivity) {
    param_name : NMOS_VTH;
    values : " 0.00005903, 0.00013996, 0.00024926, 0.00039578, 0.00076626, 0.00143066, 0.00218460, \
               0.00004536, 0.00017648, 0.00019587, 0.00043164, 0.00073766, 0.00128304, 0.00267300, \
               0.00000655, 0.00028688, 0.00021424, 0.00014293, 0.00060676, 0.00120560, 0.00256300, \
               0.00015235, 0.00000940, 0.00013138, 0.00047872, 0.00043626, 0.00106128, 0.00258060, \
               0.00013605, 0.00010076, 0.00014863,-0.00007583, 0.00093104, 0.00073040, 0.00226820, \
               0.00002886, 0.00021371, 0.00008452, 0.00039446, 0.00053548, 0.00152218, 0.00108592, \
               0.00021074, 0.00000191,-0.00001433,-0.00004354, 0.00005460, 0.00122914, 0.00221760";
  }
  rise_transition_sensitivity (slp_load_tmg_sensitivity) {
    param_name : NMOS_VTH;
    values : "-0.00000002,-0.00000033,-0.00000080,-0.00000015,-0.00000009,-0.00000013,-0.00000032, \
               0.00000040,-0.00000148,-0.00000045, 0.00000003,-0.00000024,-0.00000020,-0.00000044, \
               0.00009423, 0.00004180, 0.00003696, 0.00001804, 0.00000135,-0.00000039,-0.00000002, \
               0.00011414, 0.00014032, 0.00010179, 0.00010369, 0.00006039, 0.00000881, 0.00000013, \
               0.00009821, 0.00025454, 0.00027346, 0.00028336, 0.00024398, 0.00013671, 0.00002798, \
               0.00006888, 0.00033220, 0.00030118, 0.00052910, 0.00034628, 0.00031262, 0.00013435, \
              -0.00032978, 0.00027896, 0.00052360, 0.00069344, 0.00080300, 0.00063184, 0.00058080";
  }

...
```

```
    }

    internal_power () {
      related_pin : "A";
      fall_power(slp_load_pwr) {
        values ("0.00025, 0.00025, 0.00026, 0.00026, 0.00026, 0.00026", \
                "0.00025, 0.00025, 0.00025, 0.00026, 0.00026, 0.00026, 0.00026", \
                "0.00028, 0.00028, 0.00027, 0.00027, 0.00026, 0.00026, 0.00026", \
                "0.00041, 0.00038, 0.00038, 0.00032, 0.00020, 0.00028, 0.00028", \
                "0.00075, 0.00058, 0.00059, 0.00049, 0.00040, 0.00036, 0.00035", \
                "0.00155, 0.00122, 0.00127, 0.00103, 0.00075, 0.00062, 0.00049", \
                "0.00322, 0.00307, 0.00258, 0.00245, 0.00194, 0.00142, 0.00105");
      }
      rise_power(slp_load_pwr) {
        values ("0.00081, 0.00082, 0.00083, 0.00084, 0.00085, 0.00087, 0.00091", \
                "0.00083, 0.00083, 0.00083, 0.00084, 0.00085, 0.00087, 0.00091", \
                "0.00088, 0.00088, 0.00087, 0.00086, 0.00086, 0.00087, 0.00091", \
                "0.00102, 0.00100, 0.00098, 0.00095, 0.00092, 0.00091, 0.00093", \
                "0.00136, 0.00131, 0.00125, 0.00118, 0.00111, 0.00104, 0.00100", \
                "0.00212, 0.00203, 0.00191, 0.00177, 0.00160, 0.00144, 0.00129", \
                "0.00377, 0.00363, 0.00343, 0.00313, 0.00279, 0.00244, 0.00210");
      }
      /* Additional Sensitivity Tables for Dynamic Power */
      fall_power_sensitivity (slp_load_pwr_sensitivity) {
        param_name : NMOS_VTH;
        values : "-0.00000651,-0.00000314,-0.00000182,-0.00000097,-0.00000051,-0.00000026,-0.00000019, \
                  -0.00001711,-0.00001179,-0.00000882,-0.00000586,-0.00000341,-0.00000183,-0.00000107, \
                  -0.00006404,-0.00005019,-0.00004244,-0.00003198,-0.00002252,-0.00001430,-0.00000796, \
                  -0.00014674,-0.00012408,-0.00010868,-0.00008862,-0.00006475,-0.00004446,-0.00002727, \
                  -0.00030833,-0.00028468,-0.00026037,-0.00022770,-0.00017974,-0.00013266,-0.00008698, \
                  -0.00047267,-0.00044638,-0.00042537,-0.00037345,-0.00031603,-0.00024046,-0.00017006, \
                  -0.00080839,-0.00078749,-0.00075537,-0.00070235,-0.00061336,-0.00050006,-0.00036784";
      }
      rise_power_sensitivity (slp_load_pwr_sensitivity) {
        param_name : NMOS_VTH;
        values : "-0.00005020,-0.00005173,-0.00005377,-0.00005647,-0.00005915,-0.00006109,-0.00006223, \
                  -0.00005887,-0.00005519,-0.00005397,-0.00005392,-0.00005544,-0.00005806,-0.00006019, \
                  -0.00010746,-0.00009569,-0.00008856,-0.00007933,-0.00007048,-0.00006470,-0.00006202, \
                  -0.00018711,-0.00016786,-0.00015543,-0.00013706,-0.00011660,-0.00009611,-0.00007959, \
                  -0.00035024,-0.00032615,-0.00030701,-0.00027775,-0.00023694,-0.00019118,-0.00014619, \
                  -0.00052173,-0.00049181,-0.00046541,-0.00042779,-0.00037543,-0.00030789,-0.00023694, \
                  -0.00085954,-0.00082379,-0.00079651,-0.00074745,-0.00067078,-0.00057354,-0.00045628";
      }

    ...

    }
  }
}
```

Listing A.5 Liberty standard cell library example with extensions for process parameter sensitivities based on [5, 9]

References

1. IEEE Standard for Integrated Circuit (IC) Open Library Architecture (OLA). IEEE STD 1481-2009 pp. 1–658 (2009)
2. IEC/IEEE Delay and Power Calculation Standards - Part 3: Standard Delay Format (SDF) for the Electronic Design Process (Adoption of IEEE Std 1497-2001). IEEE STD 1497-2004 pp. 1–94 (2004)
3. Incentia: Advanced on-chip-variation timing analysis. Tech. rep., Incentia Design Systems Inc. (2007)
4. Croix, J.F., Wong, D.F.: A fast and accurate technique to optimize characterization tables for logic synthesis. In: IEEE/ACM Design Automation Conference (DAC 97), pp. 337–340 (1997)
5. Liberty technical advisory board (TAB). http://www.opensourceliberty.org
6. Synopsys, Inc.: PrimeTime User Guide, B-2008.12 edn. (2008)
7. Trihy, R.: Addressing library creation challenges from recent liberty extensions. In: 45th ACM/IEEE Design Automation Conference, 2008. DAC 2008., pp. 474 –479 (2008)
8. Nangate: Nangate 45nm Open Cell Library (2008). URL http://www.nangate.com
9. Dietrich, M., Eichler, U., Haase, J.: Digital statistical analysis using VHDL. In: Design, Automation & Test in Europe (DATE'10), pp. 1007–1010. Dresden, Germany (2010). URL http://www.date-conference.com/proceedings/PAPERS/2010/DATE10/PDFFILES/08.1_3.PDF

Appendix B
Standard Formats for Simulation Purposes

B.1 VHDL/VHDL-AMS Statistical Analysis Package

VHDL is a well-known behavioral modeling language for digital circuits and systems. It is extended to VHDL-AMS to model also analog and mixed-signal systems. In VHDL(-AMS) applications, it becomes interesting to make Monte Carlo features available where the following requirements should be fulfilled

- 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

The SAE J2748 Statistical Analysis Package [1] provides VHDL-AMS functions that can be used for describing the random behavior of parameters in a VHDL/VHDL-AMS description. At the beginning of each simulation run, the parameters are initialized using random values distributed in accordance with the associated probability density or cumulative density function (PDF or CDF respectively).

The fundamental function described by the SAE J2748 standard is a random number generator STD_UNIFORM that delivers (0, 1) distributed uniform numbers. The uniform random numbers can be transformed with respect to the required distribution function. The idea behind STD_UNIFORM is to access a random number generator for uniform values as it is given in the MATH_REAL package of the IEEE library by the UNIFORM procedure. The SEED values of this procedure UNIFORM must be handled using a global storage place. Without further requirements, this place can be a read/write position in a file. The function

M. Dietrich and J. Haase (eds.), *Process Variations and Probabilistic Integrated Circuit Design*, DOI 10.1007/978-1-4419-6621-6,

Table B.1 Pre-defined functions in the statistical analysis package

Function name	Comment
STD_UNIFORM	Delivers a random value between 0 and 1
STD_NORMAL	Delivers a N(0,1) distributed random value
NORMAL	Delivers a Gaussian distributed random value with given mean value and standard deviation described by tolerances of min/max limits
BERNOULLI	Delivers Bernoulli distributed random numbers
PDF	Delivers a random value described by a piecewise linear description of a PDF
CDF	Delivers a random value described by a piecewise linear description of a CDF

STD_UNIFORM reads the SEED values from this file and determines the next (0, 1) distributed value calling the UNIFORM procedure that also delivers new SEED values. These new SEED values are written to the file and are used during the next activation of STD_UNIFORM.

Thus, every VHDL-AMS/VHDL simulator that allows for multiple runs can be used for setting up standard Monte Carlo experiments. It is also possible to replace the file by handling the administration of global SEED places inside a simulator.

The package is named STATISTICS and compiled into the VHDL_UTILITY library.

Example:

As an example, one of the regular distribution functions implementing the normal (or Gaussian) distribution for type REAL is declared as follows:

```
impure function NORMAL (
   NOMINAL:  REAL;        -- Nominal value
   TOL:      REAL;        -- Tolerance > 0.0
   TRUNCATE: BOOLEAN := TRUE;
   ZSCORE:   REAL    := 3.0;
   MODE:     STAT_MODE_TYPE := STAT_MODE
   ) return REAL;
```

Listing B.1 Declaration of function NORMAL in the STATISTICS package

In nominal mode, the function returns the nominal value. In statistical mode, and if TRUNCATE is FALSE, the function returns random values with a normal distribution with mean $\mu = NOMINAL$ and standard deviation $\sigma = \frac{|\text{NOMINAL}| \cdot \text{TOL}}{\text{ZSCORE}}$ Thus, ZSCORE describes how many standard deviations correspond to the absolute tolerance of the parameter. The default is 3, which means that 99.7% of all random values returned by the function will be within the limits of the tolerance range. If the value of TRUNCATE is TRUE, the normal distribution is truncated to the interval defined by the bounds of the tolerance range.

The function can be used to assign a value to a parameter during instantiation:

```
library VHDL_UTILITY, SPICE2VHD;
use VHDL_UTILITY.STATISTICS.all;

...

C1:  entity SPICE2VHD.CAPACITOR(SPICE)
        generic map (C => NORMAL(NOMINAL => 1.0E-9, TOL => 0.15)
        port map (P => N1, N => N2);
```

Listing B.2 Instantiation of a capacitor with a normal distributed value

Example Package: STATISTICAL_CORRELATED

The real-valued one-dimensional function STD_UNIFORM of the package STATISTICS allows also the declaration of user-specific functions. In this way, correlated random numbers can be generated. In the following, the basic functions of a package STATISTICS_CORELATED are explained.

Function STD_NORMAL:

The function creates a real vector with correlated normal distributed random numbers. The same identifier as in the one-dimensional case can be used because VHDL allows to overload functions and use the correct one regarding its arguments. The correlation is given by the correlation matrix $\boldsymbol{P}$. The correlation coefficients of two random variables are defined by the covariance of the two variables divided by the product of their standard deviations (Pearson's correlation coefficient, see (2.27)).

The function checks whether its parameter matrix *CORR* fulfills the characteristics of a correlation matrix $\boldsymbol{P}$. The diagonal elements of the matrix must be 1 and the others between -1 and 1. The correlation matrix must be a symmetric matrix. However, the fulfillment of these requirements does not guarantee that the matrix is positive-semidefinite. This is an additional condition that has to be fulfilled by a correlation matrix. The function STD_NORMAL only supports positive-definite correlation matrices *CORR*. Therefore, the correlation matrix can be decomposed using a Cholesky decomposition:

$$\boldsymbol{P} = \boldsymbol{L} \cdot \boldsymbol{L}^T, \tag{B.1}$$

where $\boldsymbol{L}$ is a lower triangular matrix (see function CHOLESKY of the package body). Afterward, a vector $\underline{Z}$ that consists of uncorrelated $N(0,1)$ normal distributed variables is generated. The function STD_NORMAL of the STATISTICS package is used for this purpose. The multiplication of $\boldsymbol{L}$ and $\underline{Z}$ delivers the vector $\underline{X}$ that contains correlated standard normal distributed random variables

$$\underline{X} = \boldsymbol{L} \cdot \underline{Z}. \tag{B.2}$$

Function STD_UNIFORM:

The function creates a real vector with correlated uniform distributed random numbers. The correlation is given by the rank correlation matrix $CORR = \boldsymbol{P}'$. The elements of the rank correlation matrix are the Spearman's rank correlation coefficients.

The Spearman's correlation coefficient ρ' is often explained as being the Pearson's correlation coefficient between the ranked variables. Rank means an integer number that characterizes the position of a value in an ordered sequence of the values. The "winner" is the smallest value. Thus, the rank 1 is assigned to the smallest value ($rg(smallestvalue) = 1$). The rank of identical values (so-called "ties") is the mean of the ranks that could be given to the equal values. Let (x_i, y_i) be pairs of n given values then the Spearman's rank correlation coefficient can be determined by

$$\rho' = \frac{\sum\limits_{i=1}^{n} \left(rg(x_i) - m_{rg_x}\right) \cdot \left(rg(y_i) - m_{rg_y}\right)}{\sqrt{\sum\limits_{i=1}^{n} \left(rg(x_i) - m_{rg_x}\right)^2} \cdot \sqrt{\sum\limits_{i=1}^{n} \left(rg(y_i) - m_{rg_y}\right)^2}}, \tag{B.3}$$

where $m_{rg_x} = \frac{1}{n} \sum\limits_{i=1}^{n} rg(x_i)$ and $m_{rg_y} = \frac{1}{n} \sum\limits_{i=1}^{n} rg(y_i)$ are the mean values of the ranks of the x and y values, resp. The formula can be simplified if there are no tie ranks. Using $\sum\limits_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$ and $\sum\limits_{i=1}^{n} i^2 = \frac{n \cdot (n+1) \cdot (2n+1)}{6}$, we get

$$\rho' = 1 - \frac{6 \cdot \sum\limits_{i=1}^{n} \left(rg(x_i) - rg(y_i)\right)^2}{n \cdot (n^2 - 1)}. \tag{B.4}$$

For two normal distributed random variables, the following relation between the Spearman's rank correlation coefficient S' and the Pearson's correlation coefficient ρ is valid [2]

$$\rho = 2 \cdot \sin\left(\frac{\pi}{6} \cdot \rho'\right). \tag{B.5}$$

The transformation of Spearman's correlation coefficients to Pearson's correlation coefficients is done by the function STD_UNIFORM. The possible infrequent problem that the target matrix of this transformation might not be positive definite is not considered at this place. It is obvious that rank correlation does not change using a monoton increasing transformation $G : \mathbb{R} \longrightarrow \mathbb{R}$ between two random variables. Each cumulative distribution function $F : \mathbb{R} \longrightarrow [0,1] \subset \mathbb{R}$ is monotonically increasing. Thus, the rank correlation of two random variables X and Y is the same as the rank correlation of $F(X)$ and $F(Y)$. Therefore,

$$\rho'_{F(X),F(Y)} = \rho'_{X,Y}. \tag{B.6}$$

This leads to the implemented approach. The matrix $\boldsymbol{P}'$ with Spearman's rank correlation coefficients is transformed into a matrix $\boldsymbol{P}$ with Pearson's correlation

coefficients using (B.5). Afterward, a vector $\underline{X}$ of correlated standard normal distributed random variables $N(\underline{0}, \boldsymbol{P})$ is created using the function STD_NORMAL. The components of $\underline{X}$ are transformed to uniform distributed components of $\underline{Y}$ using the CDF of the standard normal distribution. That means

$$Y_i = \Phi(X_i). \tag{B.7}$$

Φ is approximated based on [3, Algorithm 26.2.17]. The mapping is realized by the function STD_NORMAL_CDF_ROUND of the package body.

```
library IEEE, VHDL_UTILITY;
use IEEE.MATH_REAL.all;
use VHDL_UTILITY.all;

package STATISTICS_CORRELATED is

--/**
-- Declaration of a real-valued matrix.
--*/
type REAL_MATRIX is array (NATURAL range <>, NATURAL range <>) of REAL;

--/**
-- Correlated standard normal distributed random numbers.
--*/

impure function STD_NORMAL (CORR : REAL_MATRIX)
return REAL_VECTOR;

--/**
-- Correlated uniform distributed random numbers.
--*/

impure function STD_UNIFORM (CORR : REAL_MATRIX)
return REAL_VECTOR;

end package STATISTICS_CORRELATED;
```

Listing B.3 Package header of STATISTCS_CORRELATED

```
package body STATISTICS_CORRELATED is

--/**
-- Cholesky decomposition.
--*/

function CHOLESKY (A : REAL_MATRIX)
return REAL_MATRIX is
    constant N   : INTEGER := A'LENGTH(1) - 1;
    variable SUM : REAL;
    variable L   : REAL_MATRIX (0 to N, 0 to N) := A;
begin

    for I in 0 to N loop
        for K in I to N loop
            SUM := L(I,K);
            for J in I-1 downto 0 loop
                SUM := SUM - L(K,J)*L(I,J);
            end loop;
            if I = K then
               if SUM <= 0.0 then
                  report "A not positive definite.";
               else
                  L(I,I) := SQRT(SUM);
               end if;
            else
               L(K,I) := SUM/L(I,I);
            end if;
        end loop;
    end loop;

    for I in 0 to N loop
        if I+1 <= N then
           for J in I+1 to N loop
               L(I, J) := 0.0;
           end loop;
        end if;
    end loop;

return L;
end function CHOLESKY;
```

```
--/**
-- Approximated real-valued CDF of standard normal distribution.
--*/

function STD_NORMAL_CDF_ROUND (X : REAL)
return REAL is
    constant A  : REAL                 := 1.0/SQRT(MATH_2_PI);
    constant B0 : REAL                 := 0.2316419;
    constant B  : REAL_VECTOR (1 to 5) := (
                                           0.319381530,
                                          -0.356563782,
                                           1.781477937,
                                          -1.821255978,
                                           1.330274429);
    variable T         : REAL;
    variable RESULT    : REAL;
begin
        T      := 1.0/(1.0 + B0*ABS(X));
        RESULT := B(5);
        for I in 4 downto 1 loop
            RESULT := RESULT*T + B(I);
    end loop;
    if  X >= 0.0 then
        RESULT := 1.0 - A*EXP(-X*X/2.0)*T*RESULT;
    else
        RESULT := A*EXP(-X*X/2.0)*T*RESULT;
    end if;
return RESULT;
end function STD_NORMAL_CDF_ROUND;


--/**
-- Correlated standard normal distributed random numbers.
--*/


impure function STD_NORMAL (CORR : REAL_MATRIX)
return REAL_VECTOR is
    constant CORR0             : INTEGER := CORR'LEFT(1);
    constant CORR1             : INTEGER := CORR'LEFT(2);
    constant N                 : INTEGER := CORR'LENGTH(1) - 1;
    variable SUM               : REAL;
    variable VALUE_1           : REAL;
    variable VALUE_2           : REAL;
    variable L                 : REAL_MATRIX (0 to N, 0 to N);
    variable STD_NORMAL_UNCORRELATED : REAL_VECTOR (0 to N);
    variable STD_NORMAL_CORRELATED   : REAL_VECTOR (0 to N);
begin
    assert N = CORR'LENGTH(2) - 1
      report "Matrix CORR not quadratic."
      severity ERROR;

    -- Special case (1-dimensional)

    if N = 0 then
       assert CORR(CORR0,CORR1) = 1.0
         report "In the one-dimensional case CORR must be 1.0"
         severity ERROR;

         STD_NORMAL_CORRELATED (0) := VHDL_UTILITY.STATISTICS.STD_NORMAL;
         return  STD_NORMAL_CORRELATED;
    end if;

    -- Test correlation matrix

    for I in 0 to N loop
        for J in 0 to I loop
            VALUE_1 := CORR (CORR0 + I, CORR1 + J);

            if I = J then
               if VALUE_1 /= 1.0 then
                 report "CORR(" & INTEGER'IMAGE (I) & ","
                                & INTEGER'IMAGE (J) & ") = "
                                & REAL'IMAGE(VALUE_1) & " unequal 1.0."
                 severity ERROR;
               end if;
            else
               if abs(VALUE_1) > 1.0 then
                 report "CORR coefficient not correct (|" & REAL'IMAGE(VALUE_1) & "| > 1.0)"
                 severity ERROR;
               end if;

               VALUE_2 := CORR (CORR0 + J, CORR1 + I);

               if VALUE_1 /= VALUE_2 then
                 report "CORR matrix not symmetric ["
                         & "CORR (" & INTEGER'IMAGE(I) & "," & INTEGER'IMAGE(J) &") = " & REAL'IMAGE(VALUE_1)
                           & " /= "
                         & "CORR (" & INTEGER'IMAGE(J) & "," & INTEGER'IMAGE(I) &") = " & REAL'IMAGE(VALUE_2)
```

```
                        & "]."
                  severity ERROR;
               end if;

            end if;
         end loop;

    end loop;

    -- Cholesky algorithm to determine lower tiangle matrix

    L := CHOLESKY (CORR);

    -- Test of result CORR = L * L^T required

    if NOW = 0.0 then
       for I in 0 to N loop
           for J in 0 to N loop
               SUM := 0.0;
               for K in 0 to N loop
                   SUM := SUM + L(I,K)*L(J,K);
               end loop;

               VALUE_1 := SUM;
               VALUE_2 := CORR(CORR0+I, CORR1+J);

               if abs(VALUE_1 - VALUE_2) > 1.0E-9 then
                  report "Difference in Cholesky results ["
                         & "L*Trans(L) (" & INTEGER'IMAGE(I) & "," & INTEGER'IMAGE(J) &") = "
                           & REAL'IMAGE(VALUE_1) & " /= "
                         & "CORR (" & INTEGER'IMAGE(I) & "," & INTEGER'IMAGE(J) &") = " & REAL'IMAGE(VALUE_2)
                         & "]."
                   severity WARNING;

                 report "Cholesky result is not correct."
                 severity WARNING;
               end if;
           end loop;
       end loop;
    end if;

    -- Uncorrelated STD normal distributed random values

    for I in 0 to N loop
        STD_NORMAL_UNCORRELATED (I) := VHDL_UTILITY.STATISTICS.STD_NORMAL;
    end loop;

    -- Correlated STD normal distributed random values

    for I in 0 to N loop
        SUM := 0.0;
        for J in 0 to I loop
            SUM := SUM + L (I,J)*STD_NORMAL_UNCORRELATED (J);
        end loop;
        STD_NORMAL_CORRELATED (I) := SUM;
    end loop;

return STD_NORMAL_CORRELATED;
end function STD_NORMAL;


--/**
-- Correlated uniform distributed random numbers.
--*/


impure function STD_UNIFORM (CORR : REAL_MATRIX)
return REAL_VECTOR is
    constant CORR0             : INTEGER := CORR'LEFT(1);
    constant CORR1             : INTEGER := CORR'LEFT(2);
    constant N                 : INTEGER := CORR'LENGTH(1) - 1;
    variable CORR_NORMAL       : REAL_MATRIX (0 to N, 0 to N);
    variable RESULT            : REAL_VECTOR (0 to N);
begin
    assert N = CORR'LENGTH(2) - 1
      report "Matrix CORR not quadratic."
      severity ERROR;

    -- Special case (1-dimensional)

    if N = 0 then
       assert CORR(CORR0,CORR1) = 1.0
         report "In the one-dimensional case CORR must be 1.0"
         severity ERROR;

         RESULT (0) := VHDL_UTILITY.STATISTICS.STD_UNIFORM;
         return RESULT;
    end if;
```

```
    -- Transformation of Spearman correlation matrix to Pearson correlation matrix
    -- Reason: Spearman correlation will be preserved   by strictly monoton transformation.

    for I in 0 to N loop
        for J in 0 to N loop
            if I = J then
               CORR_NORMAL (I,J) := CORR(CORR0+I, CORR1+J);
            else
               CORR_NORMAL (I,J) := 2.0*SIN(MATH_PI/6.0*CORR(CORR0+I, CORR1+J));
            end if;
        end loop;
    end loop;

    RESULT := STD_NORMAL (CORR_NORMAL);

    for I in 0 to N loop
        RESULT (I) := STD_NORMAL_CDF_ROUND(RESULT(I));
    end loop;

return RESULT;
end function STD_UNIFORM;

end package body STATISTICS_CORRELATED;
```

Listing B.4 Package body of STATISTICS_CORRELATED

B.2 Probalistic Distribution Functions in Verilog-AMS

Verilog-AMS is another behavioral modeling language to describe digital, analog, and mixed-signal circuits. The language provides built-in probabilistic distribution functions to describe the random behavior of parameters [4].

The function $random returns a 32-bit signed integer number each time it is called, 32-bit signed integers are between $-2.147.483.648$ and $2.147.483.647$. Thus, the generation of a random integer value between for example -100 and 100 is given by the following code fragment

```
integer rand_int;
...
rand_int = $random % 101;
```

$U(0,1)$ uniformly distributed random numbers can be generated by

```
real rand_real;
...
rand_real = ($random + 2147483648.0)/4294967295.0;
```

Verilog-AMS defines a set of predefined random number generator that can be used to initialize parameters. The functions are supported in digital and analog context. Table B.2 gives an overview on the real-valued versions of these function.

All these functions provide an additional string parameter. Its value can be "global" or "instance." This allows to characterize several instances of the same model considering global and local variations. If the value is "global," then in a Monte Carlo simulation run only one value is created that is used by different instances. If the value is "instance," then a new value is generated each instance that references the associated parameter.

The paramset statements in Verilog-AMS that is used in a similar manner as the model card in Spice-like simulation engines supports in this way the description of global and local varying technology parameters (see [4, Sect. 6.4.1] Paramsets statements).

Table B.2 Probabilistic distribution functions in Verilog-AMS

Function name Verilog-AMS	C function in Verilog HDL	Comment
$rdist_uniform	uniform	Delivers a uniformly random value in the interval from `start` and `end`
$rdist_normal	normal	Delivers a Gaussian distributed random value defined by `mean` value and `standard_deviation` (see (2.29))
$rdist_exponential	exponential	The PDF $f_X(x) = \frac{1}{\mu} \cdot e^{-\frac{x}{\mu}}$ for $x \geq 0$ (otherwise 0) is given by the `mean` value μ. See also notes on page 38. The exponential function is often used to characterize the time between failures.
$rdist_poisson	poisson	The function shall deliver the integer value k (here represented by a real number) with the probability $P(X = k) = \frac{\mu^k}{k!} \cdot e^{-\mu}$ with $k \geq 0$. It is defined by the `mean` value μ. In reliability analysis, it is often used to characterize the number of failures in a given time.
$rdist_chi_square	chi_square	The chi-square distribution is defined by its `degree_of_freedom` df. It is the distribution of the sum of squares of df independent standard $N(0,1)$ normal distributed random variables. It is widely used in statistical theory for hypothesis testing (see also page 34 and Sect. 4.7.2.2). The PDF of values less than 0 is 0.
$rdist_t	t	The Student's t distribution is defined by its `degree_of_freedom` df. It is the distribution of the quotient built up by a standard normal distributed random variable and the square root of a chi-square distributed random variable divided by the degrees of freedom (see also [5]). The Student's t distribution is used in statistical theory to describe confidence intervals for instance (see also page 57).
$rdist_erlang	erlang	The Erlang distribution is a special case of the gamma distribution. It can be used to describe queueing systems. Details can be found in [68].

References

1. SAE J 2748: VHDL-AMS Statistical Packages. Electronic Design Automation Statndards Committee (2006)
2. Fackler, P.L.: Generating correlated multidimensional variates. Online published: www4.ncsu.edu/~pfackler/randcorr.ps
3. Abramowitz, M., Stegun, I.A.: Handbook of mathematical functions with formulas, graphs, and mathematical tables. U.S. Govt. Print. Off., Washington (1964)
4. Verilog-AMS Language Reference Manual Version 2.3.1. Acellera Organization (2009)
5. Saucier, R.: Computer generation of statistical distributions. Tech. rep., Army Research Laboratory (2000). URL http://ftp.arl.mil/random/

Glossary

Analysis of variance (ANOVA) A method to detect significant differences between more than two samples (a generalization of the simple t-test).

Corner-case point A corner-case point is characterized by a combination of extreme values of parameters. Parameters or environmental conditions at their limits are applied to investigate the extreme behavior of a system.

Design flow Design flows are the explicit combination of electronic design automation tools to accomplish the design of an integrated circuit.

Design reuse The ability to reuse previously designed building blocks or cores on a chip for a new design as a means of meeting time-to-market or cost reduction goals.

Differential nonlinearity (DNL) The DNL is a measure to characterize the accuracy of a digital-to-analog converter. It is the maximum deviation between the analog values that belong to two consecutive digital input values and the ideal Least Significant Bit (LSB) value.

EDA Electronic design automation (also known as EDA or ECAD) is a category of software tools for designing electronic systems such as printed circuit boards and integrated circuits. The tools work together in a design flow that chip designers use to design and analyze entire semiconductor chips.

Generic engineering model (GEM) Description of a step-by-step circuit design creation process, considers different design views (specification, model, schematic, layout), executable within an EDA-Design-Framework, enables efficient design reuse principles for analog circuit design.

Integral nonlinearity (INL) The INL measures the maximum deviation between the actual output of a digital-to-analog-converter and the output of an ideal converter.

Intellectual property core (IP-Core) Reusable unit of logic, cell, or chip layout design that is the intellectual property of one party. IP cores are used as building blocks within chip designs.

IC layout Integrated circuit layout, also known IC layout, IC mask layout, or mask design, is the representation of an integrated circuit in terms of planar geometric

M. Dietrich and J. Haase (eds.), *Process Variations and Probabilistic Integrated Circuit Design*, DOI 10.1007/978-1-4419-6621-6,

shapes, which correspond to the patterns of metal, oxide, or semiconductor layers that make up the components of the integrated circuit.

Monte Carlo (MC) methods/Monte Carlo simulation A random experiment, applied if an analytical description of the system seems to be hardly or not possible, (simulations of integrated circuits, e.g., transistors, library cells, chips). To investigate the behavior of a performance value y of interest, a great number n of simulations have to be made, where the process parameters x_i were changed at random following a given probability distribution.

Potentiometer A potentiometer is a three-terminal resistor with a sliding contact that forms an adjustable voltage divider. If only two terminals are used (one side and the wiper), it acts as a variable resistor.

Principal component analysis (PCA) A method to reduce the complexity. It transforms a number of possibly correlated random variables into a smaller number of uncorrelated random variables. These uncorrelated variables, called principal components. They are linear combinations of the original variables.

Response surface methods (RSM) A method to find relationships between performance and process characteristics $y = h(\underline{x})$, which allows easy predictions of y for given parameter configuration $\underline{x}$.

Statistical error Statistical or random errors are caused by unknown or unpredictable changes in parameters of a system.

Safe operating area (SOA) The SOA is defined as the region of voltages and currents or power where a device can safely operate over its lifetime without self-damage. The SOA has to consider degradation of parameters.

Systematic error Systematic errors result in deviations between expected and observed results that can be predicted.

Worst-case point The worst-case point is the parameter set of highest probability density that a parametric fault occurs under worst-case operating conditions..

Worst-case analysis Worst-Case Analysis determines, for every specification separately, the most likely process parameter set (i.e., the process parameter set "closest" to the nominal process parameter set), at which the value of the performance of interest is exactly the specification value under worst-case operating conditions.

Index

M. Dietrich and J. Haase (eds.), *Process Variations and Probabilistic Integrated Circuit Design*, DOI 10.1007/978-1-4419-6621-6,