



7th RTD Framework Program

REALITY

Reliable and Variability Tolerant System-on-a-Chip Design in More-Moore Technologies

Contract No 216537



Deliverable D2.4 (Part1)

Report on Variation-Aware Statistical Information Format

Editor: Paul Zuber, Petr Dobrovolny, Miguel Miranda
Co-author / Acknowledgement:
Status - Version: V1.1
Date: 20/10/2010
Confidentiality Level: Public
ID number: IST-216537-WP2-D2.4-v1p1_part1

© Copyright by the REALITY Consortium

The REALITY Consortium consists of:

Interuniversity Microelectronics Centre (IMEC vzw)	Prime Contractor	Belgium
STMicroelectronics S.R.L. (STM)	Contractor	Italy
Universita Di Bologna (UNIBO)	Contractor	Italy
Katholieke Universiteit Leuven (KUL)	Contractor	Belgium
ARM Limited (ARM)	Contractor	United Kingdom
University Of Glasgow (UoG)	Contractor	United Kingdom



1. Disclaimer

The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

2. Acknowledgements

The editors acknowledge the contributions by Bart Dierickx.

3. Document revision history

Date	Version	Editor/Contributor	Comments
09/04/2010	V0.1	Paul Zuber	Framework
10/05/2010	V0.2	Paul Zuber	Insert imec Contribution
10/05/2010	V1.0	Petr Dobrovolny	Attached VAMIF javadoc
28/10/2010	V1.1	Miguel Miranda	Updated content in subsections: 15.8.2, 15.8.3, 15.8.4



4. Preface

The scope and objectives of the REALITY project are:

- Development of design techniques, methodologies and methods for real-time guaranteed, energy-efficient, robust and adaptive SoCs, including both digital and analogue macro-blocks“

The Technical Challenges are:

- To deal with increased static variability and static fault rates of devices and interconnects.
- To overcome increased time-dependent dynamic variability and dynamic fault rates.
- To build reliable systems out of unreliable technology while maintaining design productivity.
- To deploy design techniques that allow technology scalable energy efficient SoC systems while guaranteeing real-time performance constraints.

Focus Areas of this project are:

- “Analysis techniques” for exploring the design space, and analysis of the system in terms of performance, power and reliability of manufactured instances across a wide spectrum of operating conditions.
- “Solution techniques” which are design time and/or runtime techniques to mitigate impact of reliability issues of integrated circuits, at component, circuit, architecture and system (application software) design.

The REALITY project has started its activities in January 2008 and is planned to be completed after 30 months. It is led by Dr. Miguel Miranda of IMEC. The Project Coordinator is Dr. Miguel Miranda. Five contractors (STM, ARM, KUL, UoG, UNIBO) participate in the project. The total budget is 2.899 k€.



5. Abstract

The goal of this WP is to develop advanced methodologies and techniques for Statistical Analysis all the way from the device level to the system level. The WP also targets developing and fully characterizing a limited standard cell library (50-100 cells) for synthesis based on restricted design rules for use in WP2, WP3, WP4, and WP5. Novel techniques to percolate variability all the way from the device level to the system level shall be developed to evaluate the impact that intrinsic variability will have on timing, energy, and yield of the complete SoC architecture, including a view on the impact of application-dependent activity. Commercial EDA solutions (e.g., fast circuit simulators, SSTA tools, power analysis tools, etc) shall be reused in the flow wherever possible in combination with Monte Carlo-based simulation techniques. Also considered in this WP is the strategic aspect of the standardization of the interfaces between different abstraction levels to enable the propagation of variability specific information throughout the design flow in order to guarantee the compatibility with existing electronic design simulation/verification tools.

This document is the deliverable D 2.4 comprising a description of an electronic information format for data under process variability. This is used to ease the link between levels of a variability aware design flow such as the one set in place in overall WP2. On top of that, there is an application interface to access the data, and an application layer for graphical representation.



6. List of Abbreviations

REALITY	Reliable and Variability tolerant System-on-a-chip Design in More-Moore Technologies
PDF	Probabilistic Density Function
RTL	Register Transfer Level
SoC	System on Chip
EDA	Electronic Design Automation
SSTA	Statistical Static Timing Analysis
IP	Intellectual Property (block)
WID	Within Die Variations
CDF	Cumulative Density Functions
CPU	Central Processing Unit
MOSFET	Metal Oxide Field Effect Transistor
SRAM	Static Random Access Memory



7. List of Tables

8. List of Figures

Figure 1 Work package overview.....	10
Figure 2 Vamif Application Layer Examples.....	11
Figure 3 Injector concept.	35
Figure 4 Hot Carrier Degradation slope.....	39
Figure 5 MOSFET soft and hard breakdown.....	46
Figure 6 MOSFET soft and hard breakdown injection.....	48
Figure 7 Simplified MOSFET soft and hard breakdown injection.....	48
Figure 8 Healing property of NBTI (negative-bias temperature instability) and effectiveness of duty-cycle in controlling V_{th} shift.....	49



9. Table of contents

1. DISCLAIMER	2
2. ACKNOWLEDGEMENTS.....	2
3. DOCUMENT REVISION HISTORY	2
4. PREFACE	3
5. ABSTRACT	4
6. LIST OF ABBREVIATIONS	5
7. LIST OF TABLES	6
8. LIST OF FIGURES.....	6
9. TABLE OF CONTENTS	7
10. INTRODUCTION.....	9
11. THE ROLE OF THE INFORMATION FORMAT IN REALITY.....	10
11.1. OVERVIEW.....	10
11.2. LINK TO OTHER WORK PACKAGES	10
12. APPLICATION LAYER EXAMPLES	11
13. CHAPTERS OF THE IF.....	12
13.1. GENERAL CONCEPTS.....	12
13.2. CHAPTER TECHNOLOGY	12
13.3. CHAPTER COMPACT MODEL.....	13
13.4. CHAPTER CELL.....	14
13.5. CHAPTER DIGITAL	15
13.6. CHAPTER SYSTEM	15
14. SYNTAX.....	16
14.1. XML AND THE VAM IF CONVENTIONS.....	16
14.1.1. <i>Files and chapters</i>	16
14.1.2. <i>Units and dimensions</i>	16
14.2. GENERAL SYNTAX AND USE	16
14.3. VALUES, PARAMETERS, EMC_SETS, OBJECTS AND CONTAINERS	17
14.3.1. <i>The value element</i>	18
14.3.2. <i>The list element</i>	18
14.3.3. <i>The parameter element</i>	18
14.3.4. <i>The geometry element (correlation_geometry)</i>	19
14.3.5. <i>The EMC (Exponent Monte Carlo) set <code>emc_set</code></i>	23
14.3.6. <i>The object element</i>	27
14.3.7. <i>The container element</i>	28
14.4. USER GUIDE TO THE VAMIF API (APPLICATION PROGRAMMING INTERFACE).....	29
14.4.1. <i>Loading, parsing and (re)writing VAMIF chapters</i>	29
14.4.2. <i>Creating new VAMIF elements</i>	31
14.4.3. <i>Read and write data from object element</i>	32
15. UNDERLYING MODELS FOR VARIABILITY MECHANISMS	35
15.1. MOSFET STATIC VARIABILITY	35
15.1.1. <i>model</i>	35
15.1.2. <i>Object and parameters</i>	36



15.1.3.	<i>Scaling with W, L and others</i>	36
15.2.	TEMPLATE FORMAT FOR THE DESCRIPTION OF A VARIABILITY OR RELIABILITY MECHANISM.....	37
15.3.	HOT CARRIER DEGRADATION.....	38
15.3.1.	<i>Analytical model</i>	38
15.3.2.	<i>Parameters (and EMC table)</i>	39
15.3.3.	<i>scaling rules for design parameters</i>	40
15.3.4.	<i>Scaling rules for dynamic stress conditions</i>	40
15.4.	MANUFACTURING YIELD.....	41
15.5.	INTERCONNECT R AND C VARIABILITY.....	41
15.6.	LITHO VARIABILITY.....	45
15.7.	TDDDB ON MOSFETs.....	46
15.7.1.	<i>Piece-wise approximation model</i>	46
15.7.2.	<i>EMC table of parameters</i>	48
15.7.3.	<i>Rules for scaling W, L, T, t</i>	49
15.7.4.	<i>Rules for scaling to multiple sequential stress conditions</i>	49
15.8.	NBTI OF MOSFETs.....	49
15.8.1.	<i>Algorithmical model</i>	49
15.8.2.	<i>EMC table and parameters</i>	50
15.8.3.	<i>Scaling rules for circuit and use</i>	51
15.8.4.	<i>Scaling rule for multiple stress conditions</i>	51
15.9.	MOSFET HYSTERESIS.....	51
15.10.	VARIABILITY OF MOSFET TEMPORAL NOISE.....	51
15.10.1.	<i>MOSFET white noise</i>	51
15.10.2.	<i>MOSFET 1/f noise</i>	51
15.10.3.	<i>MOSFET RTS noise</i>	52
16.	OTHER UNDERLYING MODELS.....	54
16.1.	ACTIVITY, STRESS HISTORY ON CIRCUIT PARTS (CELLS) AND THEIR INPUTS.....	54
16.2.	BACKEND DEFINITION.....	54
16.3.	TEMPERATURE GRADIENT.....	55
16.4.	STANDARD CELL LIBRARIES.....	56
16.5.	REPRESENTATION OF NON-STANDARD CELLS: EMBEDDED MEMORIES.....	58
16.5.1.	<i>The MemoryVAM configuration contained</i>	58
16.5.2.	<i>Donuts of memories</i>	63
16.5.3.	<i>Memory cells</i>	64
16.6.	TOP LEVEL COMPONENTS HIERARCHY.....	64
16.7.	BACKANNOTATED NETLISTS OF COMPONENTS.....	65
16.8.	VARIABILITY AWARE YIELD PREDICTION.....	65
16.9.	EXAMPLE OF A CONFIGURATION CONTAINER.....	66
17.	SPECIFIC FAST MODELS FOR TOP-DOWN.....	67
17.1.	Q&D STANDARD CELL REPRESENTATION.....	67
17.2.	Q&D AREA.....	67
17.3.	ION IOFF.....	68
17.4.	CRITICAL PATH DISTRIBUTION.....	68
18.	REFERENCES.....	68
	APPENDIX A.....	68



10. Introduction

Key part to master the complexity and density of information of any variability aware design flow is a “Variability Aware Modelling information format” (VAM IF). Thus imec developed a format that considers connections between five levels of abstraction of modelling or simulation. It defines how variability information must be carried from the one level to the other.

The VAM IF is divided in 5 chapters or XML-files, corresponding to these levels of modelling.

- Chapter technology
- Chapter compact model
- Chapter cell
- Chapter digital
- Chapter system



11. The role of the information format in Reality

11.1. Overview

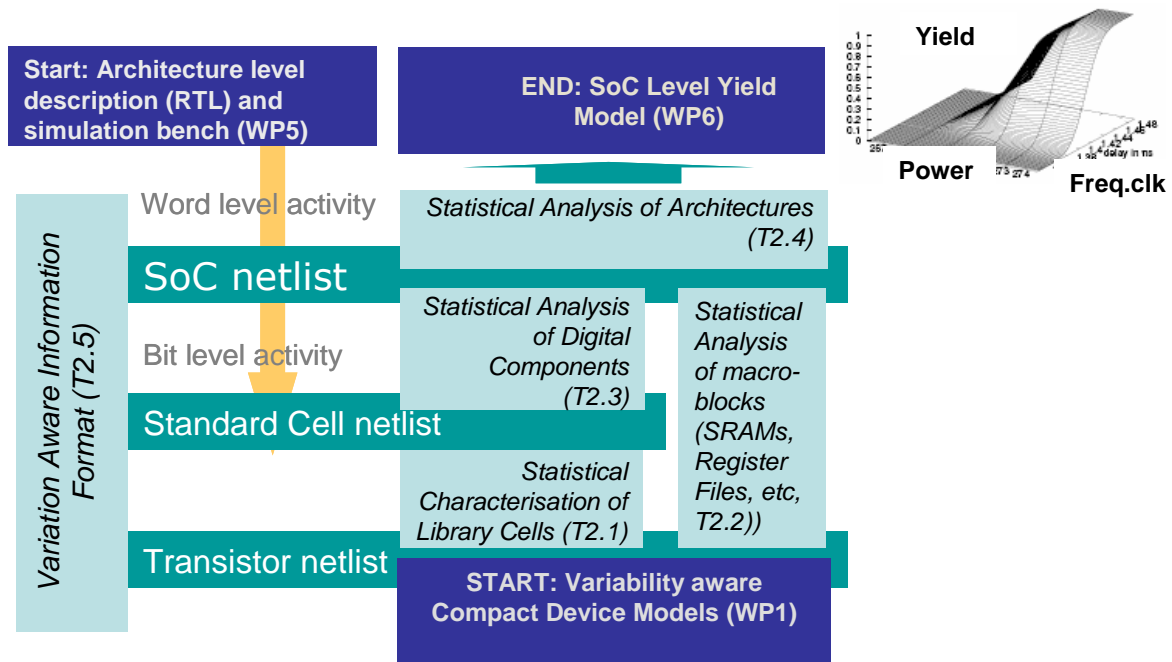


Figure 1 Work package overview.

As the Figure shows, the information format described in this deliverable plays an important role. It is positioned along and between every step of the flow, where it defines the format of information that flows between the steps. It serves as storage and retrieval system for this information.

11.2. Link to other work packages

The format accepts data from WP1 in form of injectors. These are additional circuit elements that model the variability of transistors. This can be done for several transistor types, but also differently degraded transistors.

The information format is also capable of storing and retrieving variability data on higher levels, such as the ones worked out in WP2, and integrated in WP5, like variable standard cells, macros, processor parts, or processors.

Its application layer allows to graphically publish the variability information in several formats. This is used heavily in almost all workpackages, especially WP6.



12. Application Layer Examples

The picture below shows several screen shots of the VAMIF browser application. The top-left screen shows the browser window itself on a digital chapter, highlighting a component-object (an ARM processor part). Objects can be browsed (cf Plot button top left of window).

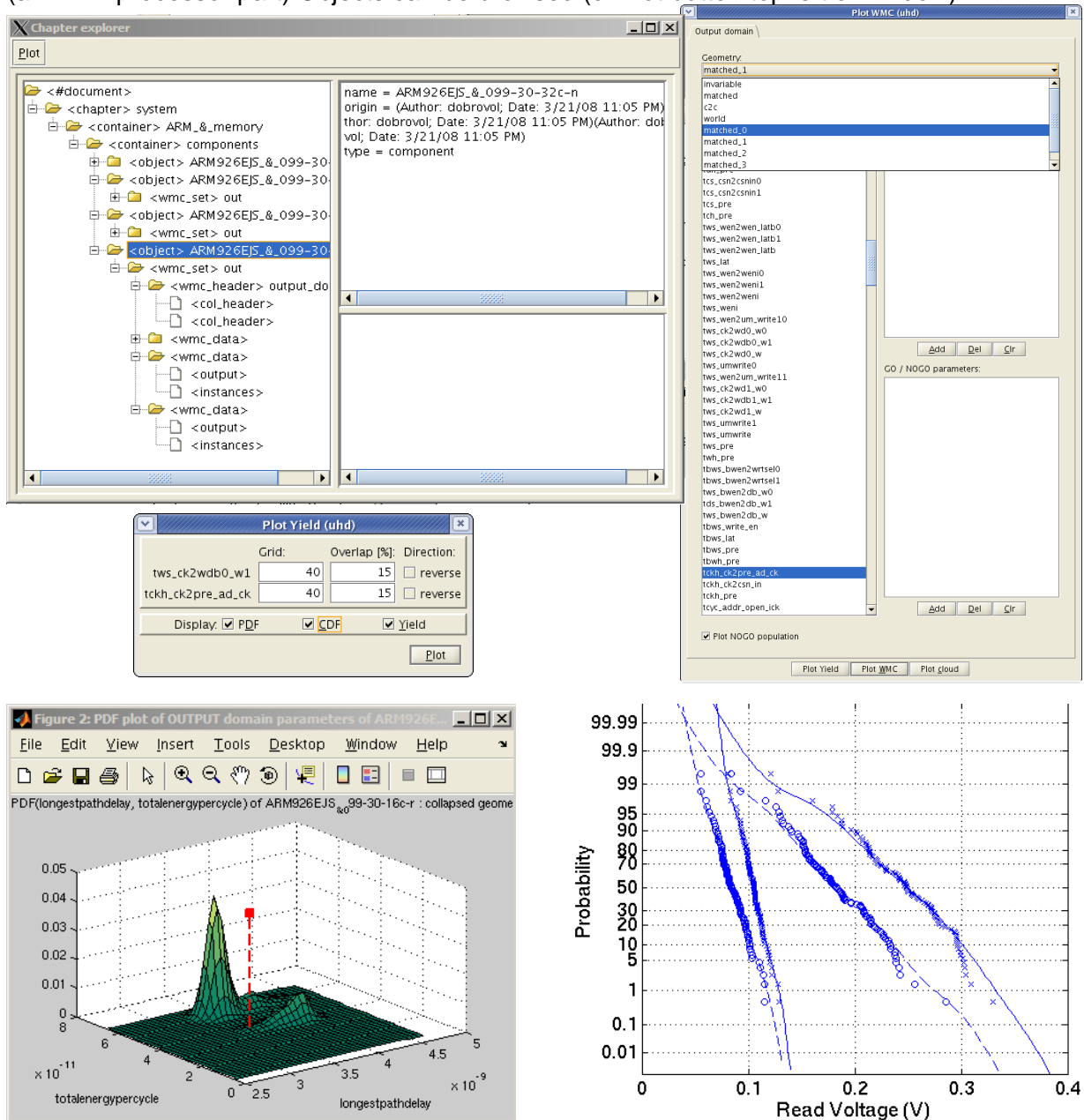


Figure 2 Vamif Application Layer Examples

To the right one finds an open plot window on a memory object. The Geometry selection box provides local (matched), global (c2c) and total (world) information and local sensitivity data (of the object to modifications in specific sub-blocks (matched_x)). The long list provides all metrics, which the user can choose to plot. There are about 500 parameters in this memory example. The user can plot pdfs, cdfs, and yield, all for 1 or 2 (compound) parameters.

The lower left plot shows an example 2-dimensional pdf of a processor part. The lower right plot shows a probit plot of the read-voltage distribution of two memories in two modes. Probit plots using this technique are used extensively to report in WP1, WP3, WP5, and WP6.



13. Chapters of the IF

13.1. General concepts

The information format is distributed over 5 “chapters” or XML-files, corresponding to the simulation levels of abstraction, having following filenames:

technology.xml
compactmodel.xml
cell.xml
digital.xml
system.xml

These 5 files are in the same directory. This set of 5 files applies to one single design (i.e. identical mask set) with one single technology option, which is reflected in the directory name.

All data in the directory is assumed to be correlated. This is a direct consequence of the principle that VAMIF is a “virtual technology” (~platform)

Scripts, programs, tools using the VAMIF should be sufficiently instructed by only giving them the path to this directory.

Scripts, programs, tools writing to a chapter should only *update* the chapter, thus leaving the unrelated data in the chapter untouched. It is forbidden that a scripts “resets” a chapter.

13.2. Chapter Technology

This chapter collects technology variability information.

The lowest level of abstraction, includes variability and reliability information from three complementary sources. Variability information is terms of dimensions or of concentrations.

⇒ the information that comes from the semiconductor fab, as measurement data sets.

As a side effect, the VAM IF thus defines the variability data that are needed from the fab

⇒ from science and literature. *Large part of the work is to devise the formulas for reliability-related effects, with the possibility to model it for deep sub micron devices.*

⇒ *TCAD simulations*



This chapter is the input for the path from technology information to compact models that can handle variability and reliability effects.

In the most trivial form, i.e. if there is no variability information at all, this chapter is empty.

Chapter **technology**

note

Tools using VAMIF should always be able to handle empty or deliberately missing information, and treat that as the absence of variability and use the nominal (or “invariable”) value (which should be available from the non variability aware design flow).

13.3. Chapter compact model

Idea: this chapter contains all information necessary for adding *electrical domain* variability and reliability information to the classical compact model (V_{th} , β ...), and other device models (R, C...) used in analog simulation and in (standard / non-standard) cell calibration. The intended users of (i.e. tools reading) this chapter are wrappers and scripts doing analog simulation, standard cell characterization, memory simulation, etc.

- Variability in this context always includes reliability, ageing and degradation effect, and wherever possible plain yield issues.
- The VAM concept considers the regular compact models as a black boxes. The compact model may as well be a physical or electrical model (i.e. a SPICE model with parameters as t_{ox} , concentrations or V_{th}), or predictive and even purely hypothetical. We strive to “*model and simulator independency*”.



In the most trivial form, i.e. if there is no variability information at all, this file and chapter may be empty. Or it may contain only a reference to the regular compact models.

```
Chapter compactmodel
```

```
chapter compactmodel
logbook
    entry 22sep2007 10:17 qwerty.exe scaling data
    entry 26sep2007 23:44 qwerty.exe scaling data

object name=mosfet // MOSFET with no variability at all
modeled
    value name=mosfet_type nmosfet_hv
    value name=path psp1.mod

object mosfet
    value mosfet_type pmosfet
    value path psp6b.mod
    parameter delta_vth
    parameter delta_beta average 0.94 stddev 0.01
    value delta_w -1e-9
    value delta_l +23e-9
    value w0 1um
    value l0 1um
```

13.4. Chapter cell

Concept: this chapter contains all information necessary for adding variability and reliability information to the classical standard cell (i.e. to simulators using standard cells), plus the variability applying to the circuits parts coming from back annotated schematics.

The intended users (readers) of this chapter are wrappers and scripts doing digital simulation, and maybe, after translation to a proper format, SSTA.

User (tool writer) challenges are:

- ⇒ How to run a library or any more complex IP unattended through cell calibration, in a reasonable time.
- ⇒ How to generate the EMC population of .lib files?
- ⇒ (in the future) How to add non-static variability (temporal noise, jitter)?
- ⇒ How to capture similar information for non-standard cells (“Macros”, such as embedded memory, mixed signal blocks...)
- ⇒ Keep non-variability corners: calibrate the library for a given set of environmental parameters (VDD, T, age).



In the most trivial form, i.e. if there is no variability information at all, this chapter contains only a reference to the regular (“invariable”) standard cell library.

```
Chapter cell
logbook
  entry 22sep2007 10:11 makeit.exe create library
  entry 26oct2007 12:12 makeit.exe create library
```

13.5. Chapter digital

Idea: this chapter contains all information necessary for adding variability and reliability information to digital simulation, architecture.

The component-level information is variability of timing, static and dynamic power. Other information is timing and application information as the activity (stress history). The Chapter is used by a tool that can estimates parametric yield. Other information that might be needed is a list of blocks that are considered as [top-level] blocks that needs calibration.

In the most trivial form, i.e. if there is no variability information at all, this chapter contains one single top level digital block. Subsequent tools (DigiVAM) will take the list of top-level blocks and calibrate them.

13.6. Chapter system

Idea: this chapter contains remaining information necessary for adding variability and reliability information tot system level simulations and yield estimation. It must also contain the top level activity information, application specifics, and external factors as temperature, VDD, temperature, age, ..

System yield is written *into* this chapter, as a set of EMC tables with total power versus clock period. Display tools as “Vamifbrowser” will plot these into the “iso yield” format.



14. Syntax

14.1. XML and the VAM IF conventions

14.1.1. Files and chapters

The VAM IF chapters are in XML format.
Each chapter is a separate file, with filename identical to chapter name.
All chapter files are in the same directory.

This VAMIF directory, and each of its 5 files is design-project and technology option specific. If one wants to explore alternate technology options, library options, architecture options, one should do that in a separate directory, possibly while linking the chapters that do not differ.

It is assumed by default that all objects (and their parameters) inside the directory are correlated.

Scripts running under VAM should use VAMIF chapters as exclusive information source or pointers thereto. The only argument to such script is the VAMIF directory path.

14.1.2. Units and dimensions

Unless otherwise noted all units are Si. E.g. 5 um are represented as 5e-6.

It is a later ¹ extension, to understand following postfixes, case sensitive:			
T	*1e12	a	/1e15
G	*1e9	p	/1e12
M	*1e6	n	/1e9
k	*1000	u	/1e6
%	/100	m	/1e3
Time is expressed in seconds. It is a not explicitly supported later extension to understand following postfixes:			
1hour	3600		
1day	86400		
1year	31557600		

14.2. General syntax and use

XML implementations of different data information are described using the following syntactic elements (related to the corresponding XML implementation itself):

Convention: All keywords have only lower case alphabetic characters [a-z], digits [0-9] and _

<code>parameter</code>	The XML element named <code>parameter</code>
<code>description</code>	The attribute named <code>description</code>
<code>double, string, ...</code>	datatypes of an attribute's or element's content
<code>(distribution)</code>	the reference to the XML element <code>distribution</code> described later
<code>(...)?</code>	the part of XML document that appears zero or one time

¹ If not implemented, a tool reading VAMIF should complain and flag such extensions.



(...)*	the part of XML document that appears zero or more times
(...)+	the part of XML document that appears one or more times
	the choice among two or more alternatives

The description of XML implementations presented in this chapter is focused on the explanation of individual elements' and attributes' structure and meaning. The precise formal descriptions of our XML applications for different chapters of the information format (from syntactic point of view) are kept in the corresponding XML Schema documents.

14.3. Values, parameters, Emc_sets, objects and containers

In the VAMIF, there are 4 basic standalone elements allowable

The

- *Container*: a generic representation of a more complex structure, itself containing zero or more values, objects and other containers
- *Value*: in most cases similar to a double, but it may be also a string etc...
- *Parameter*: this is a value with a distribution attached to it.
- *Object*: this is a group of correlated parameters (may contain also values)



14.3.1. The value element

The `value` element

<code>name=string</code>	the name attribute	required
<code>Double integer string</code>	the value	required
<code>description=string</code>	the description or comment, provided by user or program	optional
<code>Origin=string</code>	the origin of the data, either provide by users or generated by program. Format: time (YYYYMMDD_hhmm) username description. <i>to avoid data explosion: origin should NOT be automatically added to values.</i>	optional

Example in xml:

```
<value name=tox>1e-9</value>
```

14.3.2. The list element

The `list` element

<code>name=string</code>	the name attribute	required
<code>type=string</code>	The type of list	optional
<code>(Double integer string)*</code>	1 or more values	required
<code>description=string</code>	the description or comment, provided by user or program	optional
<code>Origin=string</code>	the origin of the data. <i>to avoid data explosion: origin should NOT be automatically added to values.</i>	optional

Example in xml:

```
<list type=configuration_parameter name=bpw>4 8 32</list>
```

14.3.3. The parameter element

The `parameter` element serves the user to *enter* value with attached distribution. Parameters only can be part of the VAMIF `object` element. Before parameters are used, their distributions are internally converted in an object-specific `emc_set`, for which the parameters are columns. The `avg`, `stddev`, `geometry` and distribution elements shown hereafter only serve to *enter* data. The `parameter` element, where the user wants to enter the variability in a trivial way using a simple average and standard deviation, is given as

<code>name=string</code>	the name	required
<code>description=string</code>	the description	optional
<code>Origin=string</code>	if not added, origin is automatically generated	optional
<code>avg double</code>	the average or nominal value of that parameter	Required *
<code>Stddev double</code>	the standard deviation on that average, assuming thus a gaussian distribution. If absent, zero is assumed. This distribution will become "matched", see further.	Required *

Example in xml:

```
<parameter name="delta_w" description="width variation" origin="P. Dobrovolny, 20-01-2007">
  <avg>10e-9</avg>
  <stddev>0.9e-9</stddev>
</parameter>
```

If the distribution needs to be more complex, the `parameter` element is generalized as

<code>name=string</code>	the name	required
--------------------------	----------	----------



<code>description=string</code>	the description	optional
<code>Origin=string</code>	the origin of the data if known	optional
<code>avg double</code>	the average or nominal value of that parameter	optional*
<code>Stddev double</code>	the standard deviation of that average	optional*
<code>(geometry)</code>	detailed description of a variability information	required

* if geometries are given, they supersede avg and stddev entries.

Variability information is entered in more elaborate way using geometries.

Example in XML:

```

<parameter name="tox" description="oxide thickness" origin="P. Dobrovolny, 20-01-2007">
  <avg>20.0e-9</avg>
  <stddev>1.4e-9</stddev>
  <geometry ...>
  ...
</geometry>
<geometry ...>
  ...
</geometry>
</parameter>

```

Note on the use of `parameters` and `emc_sets`
 In an `object`, two distinct ways of use of `parameters` and `emc_sets` are allowed

- every `object` has at least (and often only) the `emc_set` `name="out"`. Parameter names are the column headers of the set. This representation is used when VAM API's write objects as a whole in a chapter.
- have exactly one `parameter` element for each parameter in the `object`. Parameters distributions may refer to `emc_sets` with names differing from "out", inside the same `object`. In fact `parameter` elements are converted first to smaller `emc_sets` internally, and then all are combined into the internal `emc_set` `name="out"`. This method is likely used for manual inputting data in VAMIF.

14.3.4. The `geometry` element (correlation_geometry)

This element can only be part of a `parameter`, and has the following structure

<code>type=geo_enum</code>	the <code>correlation_geometry</code> for which the variability is considered (see description below)	required
<code>avg double</code>	the average value	optional
<code>(gauss poisson delta histogram histogram_file montecarlo montecarlo_file EMC weibull lognormal min_typ_max)+</code>	One or more distribution component (see descriptions further). Serve to "fill" a parameter distribution internally.	Optional, only used to fill in analytical distributions via VAMIF directly

Tools must anticipate handling the fact that the distributions given for each `correlation_geometry` may have different averages. E.g. in the creation of std cell or digital component delays, the resulting average for each



correlation_geometry is not a priori the same.
 Key is the “invariable” correlation_geometry, which is a reference for each.

Proposed precedence policy:
 -if a geometry and invariable is given in detail, that geometry is assumed correct.
 -if a geometry is not given, it is assumed empty – hence copied from the invariable geometry
 -if the invariable geometry itself is not given, take its value from the parameter avg if given, if not assume that invariable was zero..

Example:

```
<geometry type="w2w">
  <gauss ...>
  ...
  </gauss>
  <EMC ...>
  ...
  </EMC>
</geometry>
```

Correlation geometries

Variability is considered at 5 orthogonal levels of geometrical correlation. It means that the **type attribute** takes value from the enumeration set *geo_enum* = {"invariable", "matched", "local_systematic", "c2c", "w2w", "b2b", <some others>}:

invariable	As a reference the case without variability is tracked, in order to keep a common reference for the next 5 basic geometries. The invariable distribution is internally represented as a EMC with a single entry. For most purposes “Invariable” is very much the same as the TT corner.
matched	Or Local Random variability on close scale, applied to identical components in identical geometrical environments; it is by definition uncorrelated with local layout or inter-device distance. This geometry closely appeals to the notion “matching” or “mismatch”. Specific for matched only, and optional, are attributes correlation_distance [default infinity] and correlation_exponent [default ½]. The use of this feature is not yet defined in detail yet.
local_systematic	Local systematic (reproducible, yet unpredictable or unpredicted ²) variability on close scale due to components not being in identical layout / environments; <u>seemingly random</u> within a chip, reproducible from chip to chip.
c2c	<u>Random</u> from chip to chip, fixed within a chip, variability due to variability (which may be random or systematic) over a wafer
w2w	<u>Random</u> from wafer to wafer, within the same batch.
b2b	<u>Random</u> from batch to batch. A further refinement fab to fab (f2f) is not considered
Apart from these 1+5 basic geometries, we define following intuitively known shorthands and synonyms. (may become obsolete). These are not stored in VAMIF	

² Systematic variability that is actually predicted, thus found in back-annotation of net lists etc., is not represented in VAMIF and thus not in this number. Think of back-annotation from mask data due strains, proximity effects, OPC etc.



<code>all</code> <code>na</code>	Apply to all together: there is no geometrical distinction. In practice when writing such distribution, it goes into <code>matched</code> and the 4 other geometries are made invariable. When reading (picking) from the shorthand <code>na</code> or <code>all</code> , it should pick from all 5 geometries, and sum the values, referred to <code>invariable</code> .
<code>intradie</code> <code>ocv</code> (local)	Combines <code>matched</code> and <code>local_systematic</code> – when writing such distribution, VAMIF considers this to be equivalent to allocating all variability to <code>matched</code> (worst case) and making <code>local_systematic</code> empty.
<code>Interdie</code> (global)	Combines <code>c2c</code> , <code>w2w</code> and <code>b2b</code> – when writing such distribution, equivalent to <code>c2c</code> , where <code>w2w</code> and <code>b2b</code> become empty.

Note: we refrain from defining similar intuitive shorthands for “global”, “random” and “systematic”, “reproducible”, ... as these terms are inconsistently used in literature.

Following non-basic geometries can be stored in VAMIF too. They have normally an information function only.	
<code>userdef name=xxx</code>	user defined geometries with a name attribute. May be used for anything, e.g. to explicitly name corners.
<code>ocv</code>	Is the combined effect of <code>matched</code> and <code>local_systematic</code> . How such geometry must be constructed is subject to a specific method.
<code>world</code>	Is the combined effect of all five basic <code>correlation_geometries</code> . How such geometry must be constructed is subject to a specific method.
<code>measurement</code>	In the case that a measurement set (of a large block, or of the system) corresponding to an object exists, it can be stored here for easy comparison using the VAMIFbrowser.

How to use the `correlation_geometry` information in a Monte Carlo (-like) wrapper?

- One starts from 5 independent population in the EMC set (actually 6 as `invariable` is a singular population too). We assume that the 5 geometries are truly orthogonal (they should be, by concept!)
- *When building walls from bricks*, bricks are picked in each of the 5 geometries, with picking strategies that may differ amongst geometries.
- The resulting “wall” populations (items at the next abstraction level) have again 5 geometrical kinds of variability, represented in an internal EMC set.

Distribution components, implemented in this version of VAMIF

A `gauss` element describes Gaussian distribution using the following parameters

<code>fraction = double</code>	The fraction of this <code>gauss</code> distribution in the total population for this geometry/parameter	optional*
<code>average double</code>		required
<code>dtdev double</code>		required
<code>lower_limit double</code>	Distribution is clipped bellow this value	optional
<code>upper_limit double</code>	Distribution is clipped beyond this value	optional

¹ Sum of fractions is not necessarily 1. In fact, as distributions are internally loaded into a EMC, renormalization to 1 is done automatically. Required if there are more than one distribution component



A `uniform` element describes a uniform distribution using the following parameters

<code>fraction = double</code>	The fraction of this <code>uniform</code> distribution in the total population for this geometry/parameter	optional
<code>nof_points double</code>	Number of points used to approximate the uniform distribution. Default=10.	optional
<code>min double</code>	Distribution is clipped below this value	required
<code>max double</code>	Distribution is clipped beyond this value	required

A `emc` element refers to Weighted Monte Carlo distribution, used to enter data. It is distinct from the internal representation of the parameter in its object!

<code>fraction = double</code>	The fraction of this <code>emc</code> distribution in the total population for this geometry/parameter	Optional
<code>path string</code> <code>col_index integer</code>	path of an external ascii file, white space separated. 0th column is frequency. The column index, default [1].	optional
or <code>set_name string</code>	This refers to a <code>emc_set</code> with a different name than "out", inside the same object. Parameter names are headers of this <code>emc_set out</code> .	optional

Further distribution components, not yet implemented in this version

A `poisson` element describes Poisson distribution using the following parameters

<code>fraction = double</code>	The fraction of this <code>poisson</code> distribution in the total population for this geometry/parameter	optional
<code>average double</code>	Positive number	required
<code>factor double</code>	x-scale of distribution must be multiplied with this value	optional
<code>offset integer</code>	origin of distribution is increased with this value	optional

A `delta` element describes a simple variability specified by only one number

<code>fraction = double</code>	The fraction of this <code>delta</code> distribution in the total population for this geometry/parameter	optional
<code>Double</code>	a delta (single value) distribution	required

A `histogram` element retains histogram data as the set of number pairs

<code>fraction = double</code>	The fraction of this <code>histogram</code> distribution in the total population for this geometry/parameter	optional
<code>(double double)+</code>	a pair of numbers (thus in fact, this may also serve to input single column EMC)	required
<code>Filename string</code>		optional
<code>column_header string</code>		optional
<code>column_header string</code>		optional

A `montecarlo` element keeps a set of values from a montecarlo simulation

<code>fraction = double</code>	The fraction of this <code>montecarlo</code> distribution in the total population for this geometry/parameter	optional
<code>(double)+</code>		required
<code>path string</code>	if stored in a file	optional

Note this approach does not support correlation as the EMC does.

A `min typ max` describes simple variability specified by only three numbers.

<code>fraction = double</code>	The fraction of this <code>min typ max</code> distribution in the total population for this geometry/parameter	optional
--------------------------------	----------------------------------------------------------------------------------------------------------------	----------



<code>min double</code>		required
<code>typ double</code>		required
<code>max double</code>		required

Note: Yes indeed, this is strictly speaking not a distribution, but it is a route to introduce the classical corners into the VAM flow

A `weibull` element describes Weibull distribution parameters

A `lognormal` element describes lognormal distribution parameters

Example:

```
<gauss fraction="0.9">
  <average>20e-9</average>
  <stddev>1e-9</stddev>
</gauss>
```

14.3.5. The EMC (Exponent Monte Carlo) set `emc_set`

EMC represents by default the so-called “output domain” parameters, i.e. parameters that belong to the instantiations of the object for which the parameters are properties. E.g. the [output domain] parameters of a logic gate are delays, power, energy.

For use in RSM or binning/interpolation, “input domain parameters” are useful. These are essentially the output domain parameters of the objects used inside the object.

E.g. when creating a NAND from 4 MOSFETs, it *might* have output domain parameters:

maxdelay avdelay maxenergy avenergy staticpower

input domain parameters:

M1!delta_vth M1!delta_beta M2!delta_vth M2!delta_beta M3!delta_vth
M3!delta_beta M4!delta_vth M4!delta_beta

But preferable we use the “corrid”s of the input objects to describe the input domain parameters. See further

The element `emc_set` represents a container to keep `emc_data` elements for existing geometries of a parameter or a correlated set of parameters

`emc_set`

<code>name=string</code>	the name (the object –specific “output domain” Emc_set is named “out”)	required
<code>corrid=string</code>	Correlation id.: EMCs with the same corrid are <u>correlated</u> entry by entry. Suggestion to construct this <u>unique number</u> from machine time <code>time(0)</code> or otherwise.	Required except for matched
<code>description=string</code>	the description	optional
<code>origin=string</code>	the origin of the data if known	optional
<code>path=string</code>	the file (directory?) where EMC data are stored	optional ¹
<code>(emc_data)+</code>	one or more <code>emc_data</code> elements that keep (or point to) a numerical EMC data	required
<code>emc_header</code>	gathers <code>col_header</code> elements that identify parameters referring to the current EMC set	Optional

¹ If this attribute is omitted, the EMC data is stored directly in the VAM IF chapter

**Example:****1) the `emc_set` with EMC data directly stored in a VAMIF chapter**

```
<emc_set name="emc_set_1" origin="Author: ..." >
  <EMC_header>
    ...
  </EMC_header>
  <EMC_data ...>          ...
  </EMC_data>
  ...
</emc_set>
```

2) the `emc_set` with EMC data stored in files



```

<Emc_set name="Emc_set2" origin="Author: ... "pathname="set2.dat/">
  <EMC_header>
    ...
  </EMC_header>
  <EMC_data ... />
  <EMC_data ... />
  ...
</Emc_set>

```

The emc_header element

The emc_header element groups col_header elements that identify parameters referring to the EMC set

(<u>col_header</u>)*	zero or more <u>col_header</u> elements that identify parameters	optional
------------------------	------------------------------------------------------------------	----------

The col_header element keeps information that identify a parameter whose data are stored in corresponding column of a EMC data table

<u>index=integer</u>	the index of the column (starting from 0)	required
<u>string</u>	the name of the parameter	required

Some suggested, optional or mandatory column headers

- “entry” as such is not a column. It is the row number in the EMC table and e.g. also the returned value from PickInstanceIndex(). Row numbers start from 0.
- “ptoir” the *probability to occur in reality* is always column[0]
- “defunct”, 1 if this instance is dysfunctional, otherwise 0.
- “outlier”, 1 if this instance *contains* an outlier, otherwise 0.
- “corrid_####”: the instances in the present object are created from objects with EMC tables with a corrid (unique correlation id) being the number #### . The data in the present column are the entries (row numbers) used from that other EMC.
- “simulation_reference”: a text reference to the simulation testbench of results of the entry. Format depends on the actual script.

Example:

```

<emc_header>
  <col_header index="0">ptoir</col_header>
  <col_header index="2">dCoverC</col_header>
  <col_header index="1">dRoverR</col_header>
  ...
</emc_header>

```

The emc_data element



The `emc_data` element keeps or points to a numerical EMC data of a parameter or a set of correlated parameters for a given geometry type. The EMC data stored inside XML file are formatted in a table where each row corresponds to one data set of a population – the first value of the row is the frequency (probability), the next values correspond to concrete values of correlated set of parameters. The same format rule is applied also for data stored in a file. Because generally it can not be guaranteed that the original shape of data table will be preserved (for instance it could be lost during some XML transformations), the attribute `cols` helps to reconstruct the original shape of table.

<code>type=string</code>	the type of <code>geometry</code> to which EMC data corresponds	required
<code>pathname=string</code>	The full pathname of a file where numerical EMC data are stored	optional ¹
<code>cols=integer</code>	The number of columns in the EMC data table	required
<code>rows=integer</code>	The number of rows in EMC data table that were generated by a simulation	required
<code>(double)+</code>	The numerical EMC data	Optional

¹ If this attribute is not present, the EMC data is stored directly in VAM IF chapter. **Q: isn't this redundants wth the similar attribute in the `emc_set`?**

Example:

1) the `emc_data` element with EMC data stored directly in a VAMIF chapter

```
<emc_data type="local_systematic" cols="2" rows="100">
  1.215176571174768E-9 1.5999999999999986E-10
  3.954639285187116E-9 2.2799999999999986E-10
  1.2365241012645407E-8 2.9599999999999986E-10
  3.714723692809837E-8 3.639999999999999E-10
  1.07220707000726E-7 4.3199999999999985E-10
  2.9734390324296495E-7 4.999999999999998E-10
  7.922598189953721E-7 5.679999999999999E-10
  ...
</emc_data>
```

2) the `emc_set` element with EMC data stored in files



```
<emc_data type="w2w" cols="3" EMCs="100" pathname="D:\ic\Emc_set2\Emc_set2_w2w.dat/">
```

14.3.6. The `object` element

The `object` element contains a set of correlated [output domain] parameters and (optional) input domain parameters

<code>name=string</code>	the name of the object	required
<code>type=ObjectType_enum</code>	the type of the object ⁽¹⁾	Optional (obsolete?)
<code>description=string</code>	the description or comment, provided by user or program	optional
<code>origin=string</code>	the origin of the data. If not provided by user, the origin is automatically generated	required
<code>dir=string</code>	the directory where all data relevant to the object are stored ⁽²⁾	optional
<code>emc_set name="out"</code>	the output domain EMC set	Required if no parameters are given
<code>(emc_set) *</code>	Other emc_sets, under which the input parameter domain EMC set	optional
<code>(parameter) *</code>	Output domain parameters	only if no emc_set name="out" is given

¹ Object type, if specified, gets a value from the enumeration set `ObjectType_enum = {"cel", "component"}`. Specifying the object type enables a user/developer to exploit specific programming interface closely related to the specified type of object.

² If this attribute is not present, all related EMC data are stored directly in VAM IF chapter. If a relative path is given, this path is relative to the VAMIF chapter itself, is a separate directory.

Example:

```
<object dir="...\NAND1" name="NAND1" origin="Author: ..." type="cell">
  <Emc_set dir="...\NAND1" name="inp" origin="Author: ..." >
    <EMC_header>
      <col_header index="2"> T2_NMOS_Vth</col_header>
      <col_header index="4"> T4_PMOS_Vth</col_header>
      <col_header index="1"> T1_NMOS_Vth</col_header>
      <col_header index="3"> T3_PMOS_Vth</col_header>
    </EMC_header>
    <EMC_data type="matched" cols="5" EMCs="100" pathname="...\inp_matchd.dat"/>
    <EMC_data type="local" cols="5" EMCs="100" pathname="...\inp_local.dat"/>
    <EMC_data type="c2c" cols="5" EMCs="100" pathname="...\inp_c2c.dat"/>
    <EMC_data type="w2w" cols="5" EMCs="100" pathname="...\inp_w2w.dat"/>
    <EMC_data type="b2b" cols="5" EMCs="100" pathname="...\inp_b2b.dat"/>
  </Emc_set>
  <Emc_set dir="...\NAND1" name="out" origin="Author: ..." >
    <EMC_header>
      <col_header index="2">delay</col_header>
      <col_header index="1">power</col_header>
    </EMC_header>
    <EMC_data type="matched" cols="3" EMCs="100" pathname="...\out_matchd.dat"/>
    <EMC_data type="local" cols="3" EMCs="100" pathname="...\out_local.dat"/>
    <EMC_data type="c2c" cols="3" EMCs="100" pathname="...\out_c2c.dat"/>
    <EMC_data type="w2w" cols="3" EMCs="100" pathname="...\out_w2w.dat"/>
    <EMC_data type="b2b" cols="3" EMCs="100" pathname="...\out_b2b.dat"/>
  </Emc_set>
  <instance_list type="matched">
    NAND1_1 NAND1_2 ...
  </instance_list>
  <instance_list type="local">
    NAND1_1 NAND1_2 ...
  </instance_list>
  <instance_list type="c2c">
    NAND1_1 NAND1_2 ...
  </instance_list>
  <instance_list type="w2w">
    NAND1_1 NAND1_2 ...
  </instance_list>
  <instance_list type="b2b">
    NAND1_1 NAND1_2 ...
  </instance_list>
</object>
```



14.3.7. The `container` element

The `container` element is defined as

<code>name=string</code>	the name of the container	required
<code>type=string</code>	Following types are predefined (1) Type= <code>manufacturing</code> Type= <code>model</code> Type= <code>signature</code> Type= <code>configuration</code>	required
<code>description=string</code>	the description or comment, provided by user or program	optional
<code>origin=string</code>	the origin of the data, either provide by users or generated by program. Format: YYYYMMDD hh:mm username description	automatic
<code>(value)*</code>	Zero of more, user / case specific	optional
<code>(object)*</code>	Zero of more, user / case specific	optional
<code>(container)*</code>	Zero of more, user / case specific	optional

(1) at this moment we consider:
Containers of `type`
 ⇒ Manufacturing: this container holds information that describes the manufacturing process, design rules, ...
 ⇒ Model: this container holds models
 ⇒ Signature: this container holds “signatures”, i.e. condensed properties of unique instances of a distribution of cells.
 ⇒ Configuration: a user defined, tool specific, free format list of configuration data for tools.

In the remainder of this document, we often use following shorthand
`container xyz`
 or even
`model xyz`
 actually means:
`container type=model name=xyz [description=...] [origin=...]`
 Which looks in XML as:
`<container name=xyz type=model description=... origin=...> ... </container>`

This item may contain	value	list	paramet	emc_set	object	container
chapter					x	x
object	x	x	x	x		
container	x	x			x	x

In summary:

- an `object` contains a single [output domain] EMC set *or* one or more parameters from which the EMC set is internally created, and optional "values". EMC sets and parameters do not exist outside an object, and there is exactly one output domain



EMC set present or created in every object. (Q: is this limitation acceptable? Yes, as: correlation is guaranteed by construction and correlation propagation enforced) (one keeps the possibility to import other EMC's values via the parameter EMC attribute)

- *container's* content is free. They may contain anything, including any other containers, values and objects, but not parameters
- of these the *configuration container* is just as well free, but is not assumed to contain anything other than values and other configuration containers. Configurations are intended to store user/local/machine/tool specific things and are not for documented use in the VAMIF sense.

14.4. User guide to the VAMIF API (Application Programming Interface)

This chapter should provide concise user guide for the developers involved in the development of all simulation/ modeling levels of the overall VAM flow. The more detailed and complete description of VAM IF API will be left to Appendix chapters.

As was already mentioned, the whole input/output communication of each simulation level is carried through appropriate VAMIF chapters. Tool (wrapper) writers use the standardized interface -"VAMIF API"- to access and process data stored in a VAMIF chapter and also write back simulated (or computed) data to another (probably higher level) VAMIF chapter. The general VAMIF API functionality should be preserved over different implementation platforms. Due the platform independency and nice Matlab interface the JAVA was choose to implement VAMIF API at first. For the documentation of VAM IF API implemented in JAVA see Appendix A.

14.4.1. Loading, parsing and (re)writing VAMIF chapters

This chapter will present how to load a chapter from a file, extract and write back relevant information and again save a chapter in a file.

To load, display and write a whole chapter the class `VamifChapter` supply the set of public interface methods. The following Matlab example demonstrate their typical usage:

Example 1

```
1 techChapter = VamifChapter('technology.xml');  
2 techChapter.browse;  
3 techChapter.write('technology.xml');
```



At the first line the VAMIF chapter stored in the the file `'techchap.xml'` is loaded into the variable `techChapter`. The Matlab variable `techChapter` is actually a JAVA object with possibility to apply certain set of methods (called interface) to it. The complete public interface of a JAVA object or class can be revealed by using the Matlab command `methods`. The line 2 and 3 shows two methods of the public interface of the class `VamifChapter` – the method `display` displays the content of a chapter on the standard output and the method `write` saves a chapter in a file specified by the first argument of the method (in our example it is the same file `'technology.xml'`). Because a VAMIF chapter is in fact an XML application, the data are stored in a tree structure with several types of nodes. To access any piece of information, we have to specify a node's tree path with possible set of node's attributes to avoid ambiguity. The following Matlab example shows the way how to access VAMIF objects and values form previously loaded technology chapter (see Example 1)

Example 2

```
1 deltaW = techChapter.getVamifElement('object(name=delta_w)');  
2 thickness = techChapter.getVamifElement('/metal(name=metall)/object(name=thickness)');  
3 alfa = techChapter.getVamifElement('value(name=alfa)');
```

The argument of the method `getVamifElement` could be simply type of a node *bare name* (`parameter, object, value, ...`) or a node *tree-path name* (see line 2: `/metal(name=metall)/object(name=thickness)`) possibly combined with one or more pairs *node-attribute-name = node-attribute-value* (see line 1,2 and 3: `name=thickness, name=delta_w, ...`).

The following example demonstrates the way how modified, newly computed or simulated objects or values or any other VAMIF elements could be added into a chapter.

Example 3

```
1 techChapter.addVamifObject(EMCSet);  
2 techChapter.addVamifObject('interconnect_scaling(name=metall)', dROverR);
```



The method `addVamifElement` can have one or two arguments: at the first line the method absorbs one argument `EMCSet` - a VAMIF element that should be added to a chapter. At the second line the first argument specifies a tree-path under which the object should be added and the second argument `dROverR` is again a VAMIF element that should be added to a chapter. If the tree-path argument is missing the VAMIF element is attached directly as a child node to an object on which the method was invoked (in our example the method was invoked on `techChapter` object, so the `EMCSet` object is added at the top level of `techChapter` while `dROverR` object is added in the path specified by the second argument).

Note: Chapters being updated must be integrally locked before loading. Writing that chapter clears the lock. Read-only access remains always possible. The locking system must be foolproof (locks expire automatically). Locking is a future extension, not immediately needed.

14.4.2. Creating new VAMIF elements

Until now we discussed the situation when VAMIF elements (`object`, `container`, `value` ...) were created based on loading and parsing their XML representation from a VAMIF chapter. But the user of a VAMIF chapter also has to be able to create new VAMIF elements based for instance on the measurements data, simulation data, data from FABs and so on. The next example shows creation a new `object` element based on some hypothetical variability data to demonstrate hierarchical way of such process.

The `object` element servers for keeping correlated variability data (`parameter` elements) together with related `value` elements. The Java class `VamifObject` represents the basic way how to construct an `object` VAMIF element. Nevertheless a user instead of creating `VamifObject` most likely will use the specialized variants (Java subclasses) of the `VamifObject` called `Rule`, `Cell`, `Component`, ... The following example demonstrates the creation of `Rule` element of type `mosfet` containing some `parameter` and `value` elements.

Example 4

```
1 % create the gauss distribution 'gauss'
2 vthMean = 0.5;
3 vthSigma = 0.01;
4 gauss = GaussDist(vthMean, vthSigma);
5 gauss.setFraction(0.90);
6
7 % create the EMC distribution 'EMC'
8 EMC = EMCDist('Emc_set');
9 EMC.setFraction(0.10);
10
11 % create the matched geometry
12 geo = Geometry('matched');
13 geo.addDistComponent(gauss);
14 geo.addDistComponent(EMC);
15
16 % create the parameter Vth
17 vth = Parameter('vth', vthMean, geo);
18 vth.setDesc('The vth parameter');
19
20 NMOSTypeName = 'nmos_pt_013';
21 NMOSRule = Rule(RuleName.MOSFET);
22 NMOSRule.setAttr('type', NMOSTypeName);
23 NMOSRule.addParam(vth);
24
```



```

25 delta_w = NumericValue('delta_w', 0.0);
26 NMOSRule.addValue(delta_w);
27 delta_l = NumericValue('delta_l', 0.0);
28 NMOSRule.addValue(delta_l);
29 w0 = NumericValue('w0', 0.0);
30 NMOSRule.addValue(w0);
31 l0 = NumericValue('l0', 0.0);
32 NMOSRule.addValue(l0);
19 compChapter.addVamifElement(NMOSRule);

```

This parameter `vth` comprises real variability data only for `'matched'` geometry. This geometry has two distribution components, a gauss distribution and a EMC distribution. The lines 1-5 displays the way how to construct the gauss distribution element `gauss` from mean and sigma, at the lines 7-9 the EMC distribution element `EMC` is created and `'matched'` type of geometry `geo` (containing previously created distribution objects `gauss` and `EMC`) is constructed at lines 11-14. The parameter element `vth` is created at lines 16-18. The rule element of `'mosfet'` type is constructed at lines 20-21, the line 22 sets the `name` attribute of the rule element and the parameter element `vth` is added to the rule at the line 23. At lines 25-32 a set of value elements – `delta_w`, `delta_l`, `w0` and `l0` - are added to the rule element and finally the rule element is added to the chapter `compChapter`.

The similar approach can be utilized to create other types of VAMIF elements. The only requirement is that a VAMIF element stored in a chapter has a counterpart Java class implemented in the VAMIF API.

14.4.3. Read and write data from object element

The `object` element represents a set of correlated variability data together with some other non-variable information attached to it. For detail explanation of its structure see chapter 5.3. This chapter demonstrates how to retrieve numerical data from `object` element, how to use this data in a computation and produce newly computed `object` elements.

Example 5

```

1 % retrieve (non-variable) values
2 % from NMOSRule and PMOSRule elements
3 % -----
4 nmos_delta_w = NMOSRule.getValue('delta_w');
5 nmos_delta_l = NMOSRule.getValue('delta_l');
6 nmos_w0 = NMOSRule.getValue('w0');
7 nmos_l0 = NMOSRule.getValue('l0');
8 pmos_delta_w = PMOSRule.getValue('delta_w');
9 pmos_delta_l = PMOSRule.getValue('delta_l');
10 pmos_w0 = PMOSRule.getValue('w0');
11 pmos_l0 = PMOSRule.getValue('l0');
12
13 % create new VAMIF cell object
14 % -----
15 cellName = 'NAND2';
16 cell = Cell(cellName);
17 cell.setDir('design1/cells/NAND1');
18
19 % define input domain cell's parameters
20 % -----
21 cell.addInpParam('T1_vth');

```




```

22 cell.addInpParam('T2_vth');
23 cell.addInpParam('T3_vth');
24 cell.addInpParam('T4_vth');
25
26 % define output domain cell's parameters
27 % -----
28 cell.addOutParam('power');
29 cell.addOutParam('delay');
30
31 % set gamma as a common value for all parameters involved in the computation
32 % -----
33 VamifObject.setGamma(0.2);
34
35 % run Exponent Monte Carlo simulation over all geometries
36 % -----
37 geometry = {'matched', 'local', 'c2c', 'w2w', 'b2b'};
38 nofMCSamples = 100;
39 for j=1:length(geometry)
40   geo = geometry{j};
41   for i=1:nofMCSamples
42
43     % randomly pick Vth for each cell's transistor and apply Pelgrom rule
44     % -----
45     idx = NMOSRule.pickInstanceIdx(geo);
46     T1_prob = NMOSRule.getInstanceProb(geo, idx);
47     T1_vth0 = NMOSRule.getOutParamInstanceVal(geo, idx, 'Vth');
48     T1_vth = pelgromRule(T1_vth0, nmos_delat_w, nmos_delta_l, nmos_w0, nmos_w0);
49     idx = NMOSRule.pickInstanceIdx(geo);
50     T2_prob = NMOSRule.getInstanceProb(geo, idx);
51     T2_vth0 = NMOSRule.getOutParamInstanceVal(geo, idx, 'Vth');
52     T2_vth = pelgromRule(T2_vth0, nmos_delat_w, nmos_delta_l, nmos_w0, nmos_w0);
53     idx = PMOSRule.pickInstanceIdx(geo);
54     T3_prob = PMOSRule.getInstanceProb(geo, idx);
55     T3_vth0 = PMOSRule.getOutParamInstanceVal(geo, idx, 'Vth');
56     T3_vth = pelgromRule(T3_vth0, nmos_delat_w, nmos_delta_l, nmos_w0, nmos_w0);
57     idx = PMOSRule.pickInstanceIdx(geo);
58     T4_prob = PMOSRule.getInstanceProb(geo, idx);
59     T4_vth0 = PMOSRule.getOutParamInstanceVal(geo, idx, 'Vth');
60     T4_vth = pelgromRule(T4_vth0, nmos_delat_w, nmos_delta_l, nmos_w0, nmos_w0);
61
62     % now an analog simulation comes to determine 'delay'
63     % and 'power' for the current instance of the cell
64     cellNetlist = modifyNetlist(cell, T1_vth, T2_vth, T3_vth, T4_vth) ;
65     [delay, power] = analogSimulation(cellNetlist);
66
67     % store simulation results in the cell object
68     % -----
69     instanceName = [cellName, '_', int2str(i)];
70     prob = T1_prob * T2_prob * T3_prob * T4_prob ;
71     inpParams = [T1_vth, T2_vth, T3_vth, T4_vth] ;

```



```

72     outParams = [delay power];
73     cell.addSample(geoType, instanceName, prob, Vth, inpParams, outParams);
74 end
75 end

```

The example shows the code that enables characterizing the variability of the cell NAND2. in terms of delay and power. The input variability is represented by the variability of V_{th} of cell's transistors. We assume that the VAMIF elements NMOSRule and PMOSRule have been already loaded or created (see the code in the previous chapter).

At lines 1-11 non-variable numerical parameters used for evaluation of Pelgrom rule – `delta_w`, `delta_l`, `w0`, `l0` – are retrieved from NMOSRule and PMOSRule elements. Lines 13-17 displays the creation of the new cell VAMIF element with corresponding cell name `NAND2` and root directory where simulated results are stored. At the lines 19-29 the parameters of input and output domain are defined. The method `VamifObject.setGamma` at the line 33 sets the *Gamma* value. This method sets the *Gamma* value for all parameters involved in a computation. The set *Gamma* value remains valid till the next calling of the method `VamifObject.setGamma`. The *Gamma* is the exponent used to sample a EMC distribution. *Gamma* is a number between 0 and 1, where 0 corresponds to classic MC sampling, 1 corresponds to Entry Sampling and values in between are Weighted Monte Carlo sampling³. The outer loop over all types of variabilities contains the inner “Monte Carlo” loop. The method `pickInstanceIdx` randomly picks an index from an existing EMC distribution set. In case when a distribution of a particular variability type is not a EMC distribution, it is numerically converted to a EMC distribution. Then the method `getInstanceProb` and `getOutParamInstanceVal` return probability and concrete value of the output parameter corresponding to the randomly picked index. Retrieved V_{th} values are subject of Pelgrom rule – in this code represented by function call `pelgromRule`. Resulting V_{th} values are then used to modify original (non-variable) cell netlist to create its random instance which is then characterized by an analog simulation (lines 62-65).

The results of the cell characterization - values of `delay` and `power` - are then stored in the output domain of the cell object using the method `addSample`. After finishing the inner and outer loop the cell object `NAND2` should be complete characterize in its output domain (together with input data stored in its input domain).

³ if you do not know which gamma to choose, set gamma to 1 if you pick only one brick to build a wall, and go down to 0.2...0 if your need many independently picked bricks to build a wall. Note thus that gamma may vary significantly amongst geometries. Matched and Local_systemtic typically have values 0 ... 0.5; c2c, w2w and b2b have values around 0.5 ... 1.



15. Underlying models for variability mechanisms

This chapter contains the descriptions of variability and reliability mechanisms.

Conventions for denominating distributions used in this chapter:

V_{th} , W , T : means the nominal, average or invariable value

ΔV_{th} , ΔW , ΔT : static shifts away from the above value

If a parameter is called “delta_vth”, it means that it is referred to a certain “vth” in a relative fashion. The invariable geometry of a “delta_something” is exactly zero.

If the parameter would be called “vth” as such, the members of the EMCs are not relative.

σV_{th} , σW , σT : “distribution” of the parameter.

In case of a Gaussian distribution “ σ ” means standard deviation, but in VAM we use the symbol σ in a generalized way, implying 5 correlation_geometries etc.

Each of the 5 correlation_geometry distributions *may* be explicated as $\sigma_M V_{th}$, $\sigma_L V_{th}$, $\sigma_C V_{th}$, $\sigma_W V_{th}$, $\sigma_B V_{th}$.

Refrain from using algebra on σ that is valid for true Gaussians only.

15.1. MOSFET static variability

Part of chapter “compact model”

15.1.1. model

Most variability parameters and several degradation mechanisms are at some point during the modeling flow condensed into a ΔV_{th} and other netlist components.

For that purpose every MOSFET in a SPICE netlist is reparsed as:

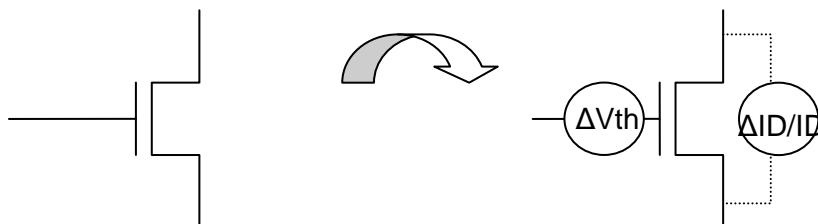


Figure 3 Injector concept.

The ΔV_{th} in this scheme is series voltage source added to the netlist; the $\Delta I_D/I_D$ ($\Delta\beta$) is a current dependent current source added to the netlist.

Note that in the VAMIF approach, the MOSFET, or any transistor, itself is a blackbox. Its *modelcard* is NOT changed. This allows using any compact model, macro, subcircuit, with minimal invasion in the existing simulation flow

Pelgrom’s rule for W and L scaling *applies to the matching geometry only*. See further

$\sigma(\Delta V_{th})_{ref}$ and $\sigma\left(\frac{\Delta I_D}{I_D}\right)_{ref}$ are the variability for the nominal transistor size,



15.1.2. Object and parameters

<p><code>object</code> <code>type=mosfet</code> <code>name=<mosfet_type></code> should correspond to the model name in SPICE (“modelcard” such as nmosfet, mypmos, nmos_hv etc.) Default: this object applies to all transistors. One may also have multiple mosfet_type entries in this container.</p>	
<p><code>value compact_model_path</code></p>	<p>path to compact model of this mosfet_type [optional].</p>
<p><code>parameter delta_vth</code></p>	<p>basic ΔV_{th} variability – note that the Pelgrom rules apply to the “matched geometry” fraction only; for other geometries the distribution is taken as such. Note also that the invariable <code>delta_vth</code> is 0</p>
<p><code>parameter delta_beta</code></p>	<p>basic $\Delta I_D/I_D$ or $\Delta\beta$ variability - note that the Pelgrom rules apply to the “matched geometry” fraction only! Note also that the invariable <code>delta_beta</code> is 0.</p>
<p><code>value delta_w</code></p>	<p><i>default = 0</i></p>
<p><code>value delta_l</code></p>	<p><i>default = 0</i></p>
<p><code>value wref</code></p>	<p><i>default = 1um, the reference MOSFET for which the ΔV_{th} and $\Delta\beta$ data are valid.</i></p>
<p><code>value lref</code></p>	<p><i>default = 1um, idem</i></p>
<p>Not yet documented, but eventually coming:</p>	
<p><code>parameter r_sd</code></p>	<p>modeling SD leakage variability</p>
<p><code>parameter r_gate</code></p>	<p>modeling gate leakage (GS, GD)</p>
<p><code>parameter r_sat</code></p>	<p>modeling saturation impedance variability</p>
<p>If documented, the <i>unpredicted</i> part of strain can be modeled</p>	
<p><code>value strainfactor</code></p>	<p>if given, additional $\Delta\beta$ variability may be added starting from overlay variability (<code>parameter alignment</code> in <code>rule litho</code>): $\Delta\beta = \text{strainfactor} * \text{alignment}$</p>

ΔV_{th} and $\Delta\beta$ in this approach are assumed to already include the variability effects of dopant and interface state fluctuations, CD variations, LER variations and layer thickness variations, and more. If one chooses to include the propagate one or more of DF, ISF, CDV, LER or LTV separately, make sure that their effect is taken out of the ΔV_{th} and $\Delta\beta$ parameters.

In such case, one could e.g. implement CD variations and LER by directly impacting the L and W parameters of the MOSFET modelcard.

Our preference and baseline however is that ΔV_{th} and $\Delta\beta$ do include all mentioned effects.

In a typical case, for each correlation_geometry, there is a EMC table representing the mentioned parameters, in a correlated fashion, e.g.:

ptoir	delta_vth	delta_beta	r_sd	r_gate
0.00013	+0.00239	-0.0445		
0.00044	-0.00097	-0.0067		
0.000012	-0.00566	+0.0125		
...		

15.1.3. Scaling with W, L and others



Pelgrom's rule for W and L scaling *applies to the matched correlation_geometry only*

$$\sigma(\Delta V_{th}) = \sigma(\Delta V_{th}_{ref}) * \frac{\sqrt{(W_{ref} + \Delta W) * (L_{ref} + \Delta L)}}{\sqrt{(W + \Delta W) * (L + \Delta L)}}$$

And a similar law applies to $\Delta I_D / I_D$:

$$\sigma\left(\frac{\Delta I_D}{I_D}\right) = \sigma\left(\frac{\Delta I_D}{I_D}\right)_{ref} * \frac{\sqrt{(W_{ref} + \Delta W) * (L_{ref} + \Delta L)}}{\sqrt{(W + \Delta W) * (L + \Delta L)}}$$

having nominal design size W_{ref} and L_{ref} . Often the nominal size is 1 μm .

For all other geometries, $\sigma(\Delta V_{th})$ and $\sigma(\Delta\beta)$ are independent of W and L!

At this moment we assume that there is no significant scaling of $\sigma(\Delta V_{th})$ and $\sigma(\Delta\beta)$ with temperature and VDS.

15.2. Template format for the description of a variability or reliability mechanism

This paragraph describes how a reliability/variability model must be setup in order to be implementable in VAMIF/VAM.

Essentially such description consists of 4 parts.

1. analytical or algorithmical model (say, C-code), containing *design values* and *technology parameters*, which describes (a) network element(s) for insertion in a SPICE netlist.

Example: in TDDDB a extra resistor R (or alike) obeys:
 $\Delta(1/R) = \Delta(t) * \text{"slope"} * \exp(A * \max(\text{abs}(VGS), \text{abs}(VDS)))$
 Where this formula contains design values t, VGS, VDS (i.e. known by the designer or simulator, technology value A (a technology dependent constant), and the parameter "slope"

2. statistical distribution of the parameters in that model

In a general case, all parameters are represented in a emc_set in a VAMIF "object". In the most elaborate case, statistics for all correlation_geometries exist, directly or indirectly derived from measurements. Minimally only geometry "matched" (= "all") must exist. If there is not variability, the EMC may contain just one entry or line.

The values are added to the object separately. Also the reference state (the conditions where the EMC measurements are taken) is given as values.

It is thus silently assumed that the EMC set refers to one single reference case for $[W_{ref}, L_{ref}, t_{ref}, T_{ref}, V_{ref}, \dots]$, and that the object thus contains also the relevant W_{ref}, L_{ref} , etc.

It is mandatory that the model contains exclusively "values" known by the designer (or designer tools), and "parameters" coming from measurements and likely subject to a distribution. Key is also: how to obtain such measurement data and transform them in a good set of parameters.

3. scaling rules allowing to translate the effect to a different case that is not in the measured set (i.e. in the set that is represented in the EMC tables). E.g. the case described in the EMC applies to W_{ref}, L_{ref} and VGS_{ref} ; now what would it transform to for a case W, L, VGS?

Examples in the following paragraphs

4. scaling the cumulative effect of multiple different stress conditions



Examples in next paragraphs

We consider worst case degradation conditions captured in a single degradation corner

These 4 items are each time a sub-paragraph in the real models here below

15.3. Hot carrier degradation

15.3.1. Analytical model

We propose to follow the approach as Chittoor Parthasarathy⁴ compiled in his PhD.

The degradation [relating to created interface states, hence to saturation current, gm, weak inversion slope and Vth] is expressed as:

$$\Delta D(t) = \left(\frac{I_D}{W * H} * \left(\frac{I_B}{I_D} \right)^m * t \right)^n$$

Where ΔD is proportional to the “damage” in terms of interface states, n is about 1/2, m is about 3 [depends on V_{GD}] and H is a technology constant.

We assume that we can calibrate the formula with a reference measurement, thus:

$$\frac{\Delta V_{th}(t)}{\Delta V_{thREF}(t_{REF})} \text{ or } \frac{\frac{\Delta g_m(t)}{g_m}}{\frac{\Delta g_{mREF}(t_{REF})}{g_m}} = \frac{\Delta D(t)}{\Delta D_{REF}(t)} = \frac{\left(\frac{I_D}{W * H} * \left(\frac{I_B}{I_D} \right)^m * t \right)^n}{\left(\frac{I_{DREF}}{W_{REF} * H} * \left(\frac{I_{BREF}}{I_{DREF}} \right)^m * t_{REF} \right)^n}$$

which opens the perspective for straightforward implementation in the VAM IF, as ΔV_{TH} and $\Delta g_m/g_m$ can be modeled with analog net list element. The variability thereof is then easily brought in via variability on these parameters themselves.

We *silently assume* that we can interchange $\Delta g_m/g_m$ and $\Delta I_D/I_D$, thus:

$$\frac{\Delta V_{th}(t)}{\Delta V_{thREF}(t_{REF})} \text{ or } \frac{\frac{\Delta I_D(t)}{I_D}}{\frac{\Delta I_{DREF}(t_{REF})}{I_D}} = \frac{\left(\frac{I_D}{W} * \left(\frac{I_B}{I_D} \right)^m * t \right)^n}{\left(\frac{I_{DREF}}{W_{REF}} * \left(\frac{I_{BREF}}{I_{DREF}} \right)^m * t_{REF} \right)^n}$$

The degradation under bias conditions that have been different during different time spans t_1, t_2, \dots is straightforward extension:

$$\frac{\Delta V_{th}(t)}{\Delta V_{thREF}(t_{REF})} \text{ or } \frac{\frac{\Delta I_D(t)}{I_D}}{\frac{\Delta I_{DREF}(t_{REF})}{I_D}} = \frac{\left(\frac{I_{D1}}{W} * \left(\frac{I_{B1}}{I_{D1}} \right)^m * t_1 + \frac{I_{D2}}{W} * \left(\frac{I_{B2}}{I_{D2}} \right)^m * t_2 + \dots \right)^n}{\left(\frac{I_{DREF}}{W_{REF}} * \left(\frac{I_{BREF}}{I_{DREF}} \right)^m * t_{REF} \right)^n}$$

⁴ C. Parthasarathy, “Etude de la fiabilité de technologies CMOS avancées: applications a la simulation de la fiabilité de conception de circuit numeriques et analogiques”, PHD thesis, 9 Oct 2006, chapter 4.2. With acknowledgements for Guido Groeseneken.



This formula is usable as such when I_B or I_B/I_D is known by the simulator. Often this is not the case.

In order to tackle that situation, we include in the rule Parthasarathy's (with some pragmatic simplification) approach to calculate I_B/I_D :

$$I_B = \frac{A_i}{B_i} * E_m * l_c * I_D * \exp\left(-\frac{B_i}{E_m}\right)$$

Where

$$E_m = \frac{V_{DS} - V_{DSAT}}{l_c} \quad \text{and} \quad V_{DSAT} \cong V_{GS} - V_{TH}$$

Hence

$$\frac{I_B}{I_D} = \frac{A_i}{B_i} * (V_{DS} - V_{GS} + V_{TH}) * \exp\left(-\frac{B_i * l_c}{V_{DS} - V_{GS} + V_{TH}}\right)$$

for $V_{DS} > V_{GS} - V_{TH}$, (for an NMOSFET). Otherwise I_B is just zero.

And we reduce to two technology constants A and V_B :

$$\frac{I_B}{I_D} = A * (V_{DS} - V_{GS} + V_{THref}) * \exp\left(-\frac{V_B}{V_{DS} - V_{GS} + V_{THref}}\right)$$

$$\log\left(\frac{\left(\frac{I_B}{I_D}\right)}{A * (V_{DS} - V_{GS} + V_{THref})}\right) = -\frac{V_B}{V_{DS} - V_{GS} + V_{THref}}$$

A and V_B are approximately constant for a given type of MOSFET in a given technology. A is not of importance as it is eliminated in the HCD formula. V_B must be obtained from a I_B/I_D calibration measurement plotted as follows. Temperature dependence is neglected.

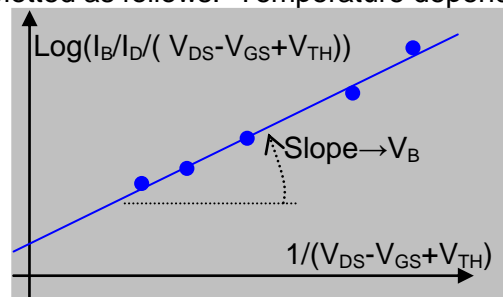


Figure 4 Hot Carrier Degradation slope.

Although conceived for NMOS, the overall HCD formula might be applied to PMOS. In that case, $V_{DS} > V_{GS} - V_{TH}$! In a PMOS the absolute value of V_{TH} decreases due to HCD, and the g_m decreases. Typically one chooses to neglect the effect of HCD in PMOS.

15.3.2. Parameters (and EMC table)

<pre>object type=hcd name=<mosfet_name></pre>	<p>the transistor type to which this applies. Default: all transistor types.</p>
<pre>parameter delta_vth_ref parameter delta_beta_ref value t_ref value id_ref value ib_ref // if not</pre>	<p>$\sigma(\Delta V_{THref}(t_{REF}))$ //</p> <p>$\sigma\left(\frac{\Delta I_{Dref}(t_{REF})}{I_D}\right)$</p> <p>in seconds!</p>



<pre> value w_ref value n value m value vb value vth_ref </pre>	<p>given, the I_B/I_D estimation formula is used</p> <p>optional, default 0.5</p> <p>optional, default 3</p> <p>// used only for estimating I_B/I_D, see formula. <i>Good default wanted – in absence, without any guarantee, use 1V.</i></p> <p>// used only for estimating I_B/I_D // default: use the MOSFET's V_{th}, something like 0.5V</p>
------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

15.3.3. scaling rules for design parameters

$$\frac{\Delta V_{th}(t)}{\Delta V_{thREF}(t_{REF})} \text{ or } \frac{\frac{\Delta I_D(t)}{I_D}}{\frac{\Delta I_{DREF}(t_{REF})}{I_D}} = \frac{\left(\frac{I_D}{W} * \left(\frac{I_B}{I_D} \right)^m * t \right)^n}{\left(\frac{I_{DREF}}{W_{REF}} * \left(\frac{I_{BREF}}{I_{DREF}} \right)^m * t_{REF} \right)^n}$$

Allows to scale for different W. L is not explicit in this formula, I_D/I_B depends on L, it is. One must of course know I_D and I_B either from the simulations or from

$$\log \left(\frac{\left(\frac{I_B}{I_D} \right)}{A * (V_{DS} - V_{GS} + V_{THref})} \right) = - \frac{V_B}{V_{DS} - V_{GS} + V_{THref}} \text{ (see above)}$$

15.3.4. Scaling rules for dynamic stress conditions



$$\frac{\Delta V_{th}(t)}{\Delta V_{th_{REF}}(t_{REF})} \text{ or } \frac{\frac{\Delta I_D(t)}{I_D}}{\frac{\Delta I_{D_{REF}}(t_{REF})}{I_D}} = \frac{\left(\frac{I_{D1}}{W} * \left(\frac{I_{B1}}{I_{D1}} \right)^m * t_1 + \frac{I_{D2}}{W} * \left(\frac{I_{B2}}{I_{D2}} \right)^m * t_2 + \dots \right)^n}{\left(\frac{I_{D_{REF}}}{W_{REF}} * \left(\frac{I_{B_{REF}}}{I_{D_{REF}}} \right)^m * t_{REF} \right)^n}$$

indicates that

the stress histories can be added before taking the result to the power n.

15.4. Manufacturing yield

This item describes hard defects that occur in manufacturing and that are not modeled via any variability and reliability rules in VAMIF. This is often described as functional yield in contrast to parametric yield.

Formula: $yield = (yield_{1m^2})^{area[m^2]}$

Where $yield_{1m^2} = \exp(-defect_density)$ Poisson: probability to have no defect in 1 m²

In [chapter system](#):

```
Container system_properties
  value chipsize nnn          in m2! (Si units)
  value chipdiagonal nnn     in m (Si units) // optional
```

In [chapter technology](#):

```
object name=manufacturing_yield
  parameter defect_density average=ddd defects per m2! (Si units)
  parameter defect_size ...      optional
```

Defect_density given as such only as an average and no distribution assumes that the statistics are pure poisson irrespective of geometry

This is an exception to the prescribed use of [parameter](#). In fact one can *imagine* that the above is a shorthand for:

```
Container type=rule name=manufacturing_yield
  parameter defects_density
    average ddd stddev ddd
    geometry na
    distribution poisson average ddd
  [parameter defect_size ... ]
```

Parameter (hence distribution) [defect_size](#) is not documented at the moment. In the future this distribution may be used for finer assessment of impact of manufacturing defects on subcircuit size.

15.5. Interconnect R and C variability

As these are device electrical parameters, this is part of chapter “compact_model”

For an interconnect resistance

$$\frac{R}{R_{av}} = \frac{R_{av} + \Delta R}{R_{av}}$$



Where R_{av} is the average [nominal, design]⁵ resistance of a resistor or interconnect, R is the actual value, ΔR is the difference between the two.

This ratio is subject to a distribution, in shorthand⁶

$$\sigma\left(\frac{R}{R_{av}}\right) = \sigma\left(\frac{R_{av} + \Delta R}{R_{av}}\right)$$

For a particular resistor or interconnect in a certain layer, use following Pelgrom-like relation for the **matched** geometry only:

$$\sigma\left(\frac{\Delta R}{R}\right) = \sigma\left(\frac{\Delta R}{R}\right)_o * \frac{\sqrt{(W_0 + \Delta W) * (L_0)}}{\sqrt{(W + \Delta W) * (L)}}$$

(W are design widths, $W + \Delta W$ are effective/electrical widths)

Similarly

$$\frac{C}{C_{av}} = \frac{C_{av} + \Delta C}{C_{av}}$$

Where C_{av} is and average [nominal, design]⁷ inter-interconnect capacitance, C is the actual value, ΔC is the difference between the two.

This ratio is subject to a distribution, in shorthand

$$\sigma\left(\frac{C}{C_{av}}\right) = \sigma\left(\frac{C_{av} + \Delta C}{C_{av}}\right)$$

For a particular later capacitance *within* a layer, use following Pelgrom-like relation for the **matched** geometry only:

$$\sigma\left(\frac{\Delta C}{C}\right) = \sigma\left(\frac{\Delta C}{C}\right)_o * \frac{\sqrt{(S_0 + \Delta S) * (L_0)}}{\sqrt{(S + \Delta S) * (L)}}$$

L is the wire length, $S + \Delta S$ are effective/electrical spacings

Within the same layer, $\Delta S = -\Delta W$

For capacitances between interconnects on *different* layers, the key layer name is the dielectric name.

$$\sigma\left(\frac{\Delta C}{C}\right) = \sigma\left(\frac{\Delta C}{C}\right)_o * \frac{\sqrt{(W_0 + \Delta W) * (L_0)}}{\sqrt{(W + \Delta W) * (L)}}$$

An in more elaborate form:

$$\sigma\left(\frac{\Delta C}{C}\right) = \sqrt{\sigma^2\left(\frac{\Delta C}{C}\right)_o + \eta \cdot \sigma^2\left(\frac{\Delta C}{C}\right)_{overlay}} * \frac{\sqrt{(W_0 + \Delta W) * (L_0)}}{\sqrt{(W + \Delta W) * (L)}}$$

L is the effective wire length, the length over which the line segments overlap

$W + \Delta W$ are effective overlap width

“overlay”: $\sigma\left(\frac{\Delta C}{C}\right)$ has a fraction due normal C and to overlay variability. The overlay

part may be significantly larger than the classic C variability. The additional fraction of “overlay” is defined by the device-specific value η (eta).

η (eta) must be obtained by the designer or by back annotation.

⁵ all three mean the same: it is the value that the simulation tool assumes in the non-variability-aware simulation flow, and to which VAMIF is assumed to add variability.

⁶ Shorthand σ stands for any distribution, including multiple geometrical correlations. It implies the same information as in reserved keyword **parameter**.

⁷ All three mean the same: it is the value that the simulation tool assumes, and to which VAMIF is assumed to add variability.



Typically:

- If one of the layers overlaps completely the other, over the full length, as in matched capacitor design, eta is zero.
- A wire or plate that orthogonally crosses: eta is zero.
- If the capacitor is partly overlapping for both metals, eta is maximal, 1.
- For two wires of same width on top of each other, eta is small, in the order of 0.0 to 0.2.

Default eta is 0.

<pre>Object type=interconnect_rc name =<layername></pre>	<p>Layername default: if not indicated, applies to all interconnect layers – must comply to interconnect name used in <code>backend_definition</code> in chapter technology. It means also that multiple versions of this rule may exist for the various metals.</p>
<pre>parameter delta_rr .</pre>	<p>basic $\sigma\left(\frac{\Delta R}{R}\right)_o$ variability – note that the Pelgrom rules apply to the “matched distribution” fraction only</p>
<pre>value l0 value w0</pre>	<p><i>Both default = minimum width (!) in that layer</i></p>
<pre>value delta_w</pre>	<p><i>default = 0. defines electrical width wrt design width. Note that delta_w equals also $-\Delta S$ (?). includes the LER effect (how?)</i></p>
<pre>value sheet_r</pre>	<p><i>sheet resistance R_{\square} for large squares. Optional, only given if known,</i></p>
<pre>value s0</pre>	<p><i>default = minimum spacing in that layer (for C)</i></p>

How should software tools proceed when they have only a backannotated R, but no clue on W and L of line segments?
 We propose: estimate L as: $L = R / R_{\square} * (W_0 + \Delta W)$, i.e. we assume that the wire has minimum width, which is a reasonable worst case. L should not drop below minimum.
 <<<Similar approach for C to be explicated??>>>

Alternative approach to import a few critical resistors / capacitors in VACCinate:
 -think of resistor naming as `R_METAL1_W2u somenode othernote bwp*500hm`
 -or code in in extra comment line in spice netlist
 -or in separate table with the resistor name as entry
 -Q: how about length parameterization



Proposition to define C from inside a spice netlist for VACCinate lateral
`c_metal1_l200u_s2u line1 line2 50fF`
 vertical
`c_diel2_l200u_w2u_eta0.1 line2 vdd 40fF`
 Any simple capacitor is perhaps subdivided in many individual capacitors. Is that worth the effort?

<pre>Object type=vertical_capacitor name =<dielectriclayername></pre>	Layername default: if not indicated, applies to all <u>dielectric</u> layers – must comply to interconnect name used in <u>backend_definition</u> in chapter technology.
<pre>parameter delta_cc_zero . parameter delta_cc_overlay .</pre>	$\sigma\left(\frac{\Delta C}{C}\right)_o$ and $\sigma\left(\frac{\Delta C}{C}\right)_{overlay}$ variability – note that the Pelgrom rules apply to the “matched distribution” fraction only
<pre>value l0 value w0</pre>	
<pre>value delta_w</pre>	<i>default = 0.</i> defines electrical width wrt design width. includes the LER effect (how?)

<pre>Object type=via name =<layername></pre>	Layername default: applies to all via and contact layers – must comply to via layer used in <u>backend_definition</u> in chapter technology.
<pre>parameter delta_rr value r</pre>	basic $\sigma\left(\frac{\Delta R}{R}\right)_o$ variability <i>nominal r of the via (optional data)</i>

A point tool will translate chapter technology to chapter compact model, and do:

$$\sigma(\Delta R/R) = \alpha * (\sigma_{thickness}/thickness) + \beta * \sigma(LER) / w + \gamma * (\text{effect of barrier layer thickness horizontal}) + \delta * (\text{effect of barrier layer thickness vertical})$$

alpha and beta are constants that need separate calculation. In the absence of any better value, we use the default 1.

thickness is the average of thickness found in the backend_definition

othickness is the distribution thereof

the value $w = W_{electrical} = W_{design} + \delta * W_{electrical}$

$\sigma(\delta * w)$ is the distribution of $\delta * W_{electrical}$, which is derived from $\delta * W_{physical}$ see below.

(both othickness and $\sigma(\delta * w_{physical})$ are subject to 5 types of geometry if available from fab.)



$\Delta w_{\text{electrical}} = f(\text{LER, technology things}) + f(\Delta w_{\text{physical}})$
 default rule is: $\Delta w_{\text{electrical}} = \Delta w_{\text{physical}}$
 $\Delta w_{\text{physical}}$ is found in the [backend_definition](#)
 LER is found in [rule litho](#)

15.6. Litho variability

Part of chapter technology.

The 4 elementary litho [parameters](#) are:

- ⇒ $\Delta \text{dose}/\text{dose}$, as derived from variability on a dose in [mJ/cm²]
- ⇒ Focus [nm]
- ⇒ Alignment [nm], overlay error
- ⇒ LER [nm], accompanied by the correlation length along edge

This litho LER may translate to the LER-effect in interconnects?>

The two first have a rather well documented impact on printing accuracy, as:

$$\text{CD} = f(\text{dose, focus})$$

A good, simple first order formula exists:

$$\text{CD} = a * \Delta \text{dose}/\text{dose} + b * \text{focus}^2 \quad [^8]$$

This may be the basis to for a rule estimating linewidth and spacing variations for interconnects and MOSFET W and L.

These a and b are functions of local layout geometry,

- ⇒ Hence, a and b are parameters with only local_systematic variability
- ⇒ Eventually, OPC-like tools may predict these, thus yielding a and b per polygon piece.

We assume that this is not the case, hence:

For the moment we propose the following approach:

- ⇒ a and b are defined as parameters (local_systematic only⁹) representing the “general layout style” used in the design. For a so called litho-friendly (RDR) layout style, a and b are lower and have smaller distributions than for a spaghetti style layout.

⁸ Staf Verhaegen, IMEC, 22-jul-07

⁹ Although a and b are represented as local_systematic, tools should apply it to all geometries of Δdose and focus in the formula $\text{CD} = a * \Delta \text{dose}/\text{dose} + b * \text{focus}$.



<code>object name=litho</code>	
<code>parameter a</code>	(local_systematic only) // reflects layout style
<code>parameter b</code>	(local_systematic only) // reflects layout style
<code>parameter delta_dose</code>	Δ dose/dose, average is 0
<code>parameter focus</code>	in [m] average is 0
<code>parameter alignment</code>	in [m] average is 0 (overlay alignment error)
<code>parameter ler</code>	in [m] line edge roughness
<code>value ler_length</code>	[m] correlation length of LER

typical values for 65nm (no guarantee on these values!)
 1sigma(focus)=50nm
 1sigma(Δ dose/dose)=2%
 1sigma(alignment)=10nm
 1sigma(LER)=2nm, averaged
 1sigma(CD) due to dose = 4%, for long parallel lines (matched)
 1sigma(CD) due to focus = 4%, for long parallel lines (matched)

15.7. TDD on MOSFETs

15.7.1. Piece-wise approximation model

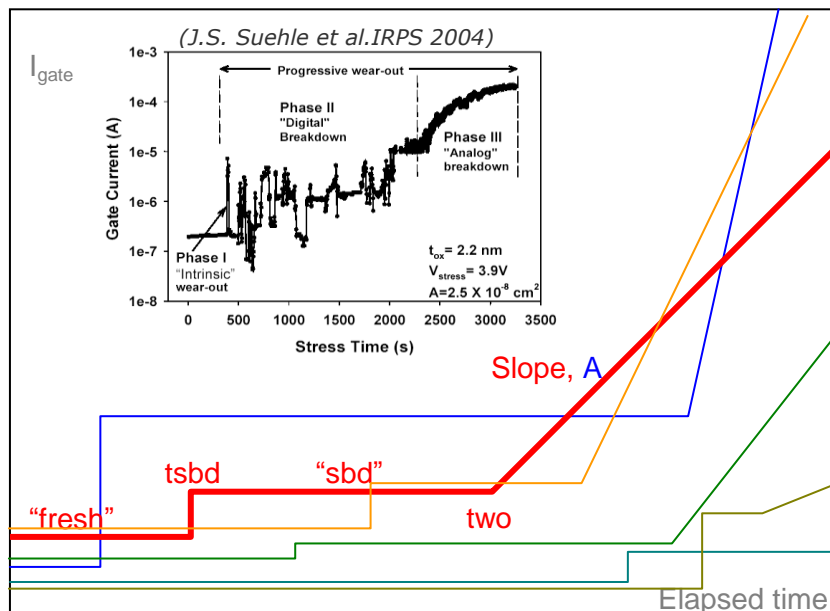


Figure 5 MOSFET soft and hard breakdown.

- soft breakdown (sbd) happens after a time **tsbd**. After that the initial “fresh” I/V characteristic changes to a “sbd” I/V characteristic. This I/V characteristic is strongly non-linear, and is characterized by.
- wear-out (wo) happens after a time **two**, after which the device goes in “hard breakdown” (hbd). The I/V behavior becomes that of a simple time dependent, progressively decreasing resistance R, obeying:
 - $\Delta(1/R) = \Delta(t) * slope * exp(A * \max(\text{abs}(V_{GS}), \text{abs}(V_{DS}))$



- Asymptotically R evolves to zero! The observed current limitation is due to external series resistances to the gate.



How is this injected (VACCinate) a Spice net list?

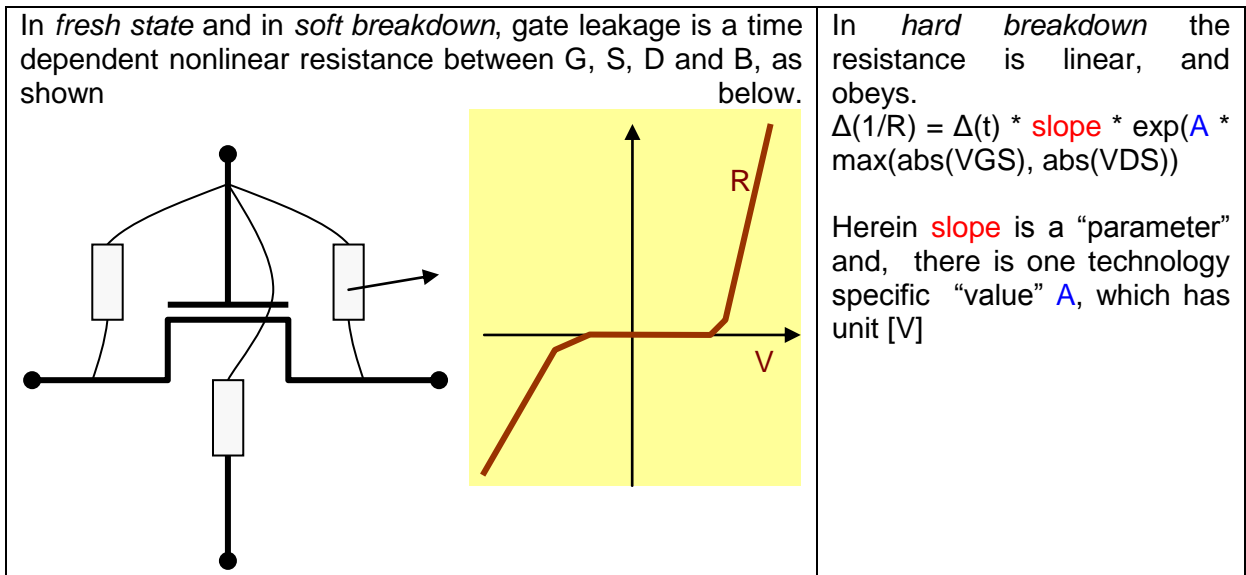


Figure 6 MOSFET soft and hard breakdown injection.

We simplify the SBD model further: we disregard the small asymmetry between inversion and accumulation, and we completely disregard the leakage to Bulk. The model will be implemented as a Verilog-A model, which is identical in both quadrants.

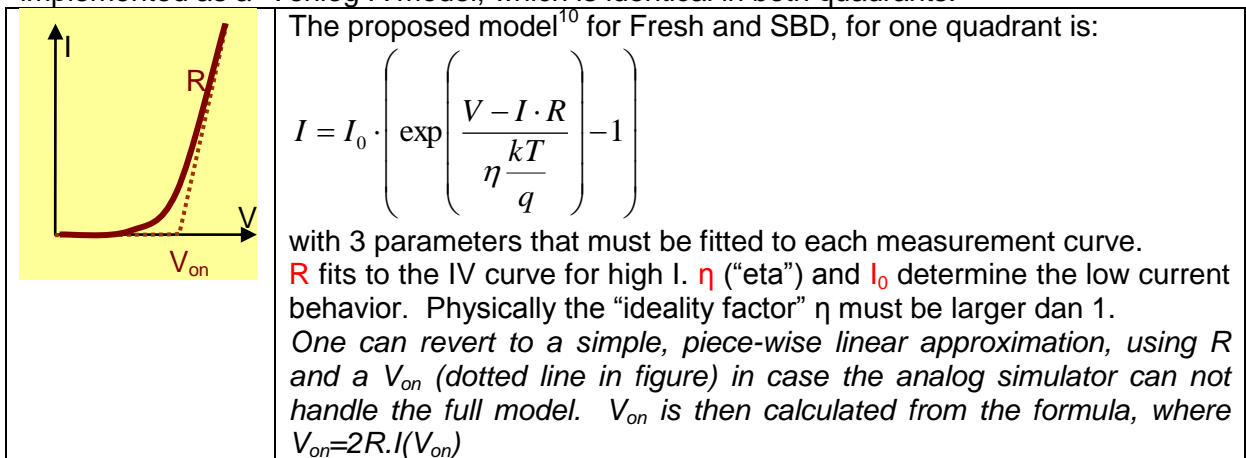


Figure 7 Simplified MOSFET soft and hard breakdown injection.

15.7.2. EMC table of parameters

This table contains a populations of measured devices (or created devices). Some parameters are given, duplicated for G-S and G-D.

These are for a given [technology, NMOS/PMOS, tox,] W, L, T, t, ..., example:

ptoir	Tsbd	two	slope_s	slope_d	rfresh	rsbd_s	rsbd_d	etafresh	etasbd_s	etasbd_d	iofresh	iosbd_s	iosbd_d
1	2022	23456	23.3	22.1	1.23e9	3.4e7	3.4e7	3.7	1.3	1.3	1.23e-19	3.4e-7	3.4e-7
1	678	10987	24.5	23.7	9.88e8	1.1e7	1.1e7	3.5	1.6	1.6	9.88e-18	1.1e-7	1.1e-7
1	13900	19765	21.7	19.0	7.88e8	2.9e7	2.9e7	3.2	1.5	1.5	7.88e-18	2.9e-7	2.9e-7

¹⁰ R. Fernández, J. Martín-Martínez, R. Rodríguez, M.Nafria, and X. H. Aymerich, “Gate Oxide Wear-Out and Breakdown Effects on the Performance of Analog and Digital Circuits”, IEEE Transactions on ED, Vol. 55-4, p.997 (2008)

In this table separate values for source and drain are given. We might consider to simply further and make these identical.

15.7.3. Rules for scaling W, L, T, t...

Should tell us “what is the effect on elapsed time” when stress condition V, T etc change. (attention: the measurement conditions after stress do not change)

A Pelgrom rule applies to R, only for Matched

15.7.4. Rules for scaling to multiple sequential stress conditions

Approach: each stress condition on its own creates an “elapsed time”. The different stress conditions just accumulate those times.

15.8. NBTI of MOSFETs

15.8.1. Algorithmical model

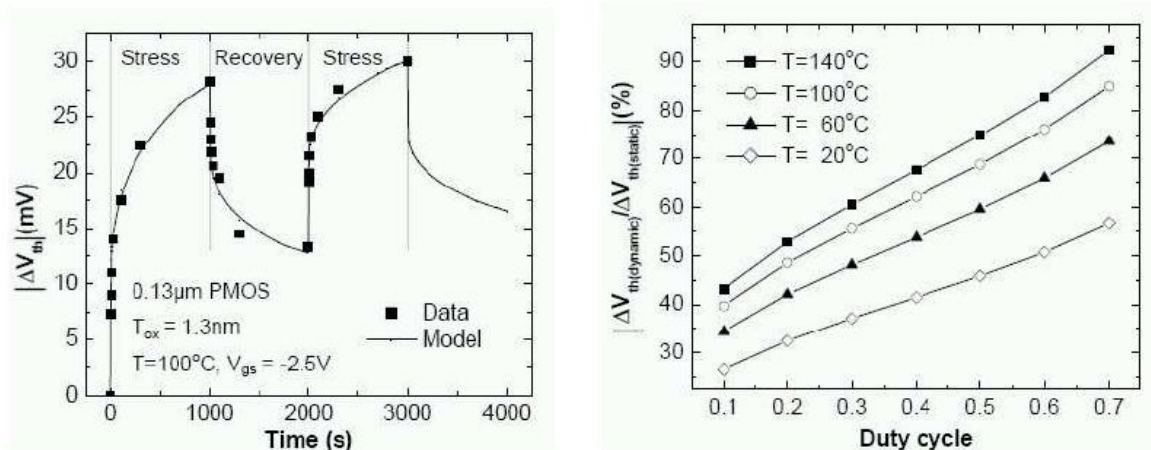
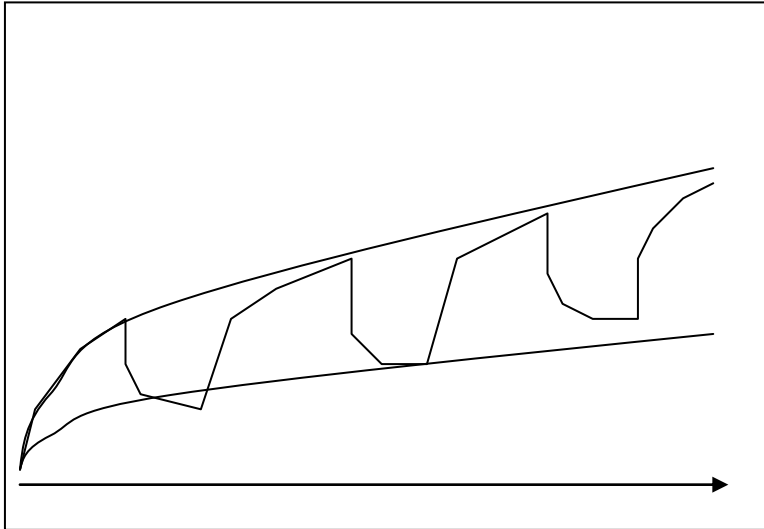


Figure 8 Healing property of NBTI (negative-bias temperature instability) and effectiveness of duty-cycle in controlling V_{th} shift¹¹

¹¹ Vattikonda R., Wang W., Cao Y., “Modeling and Minimization of PMOS NBTI Effect for Robust Nanometer Design”, DAC 2006



Function



NBTI creates an upper and lower boundary between which the transistor will move. The actual value at evaluation time is not known (pick randomly???)

$$\Delta V_{th_{min}} = f(\text{time, duty_cycle, toggle_rate, VGS, VDS, ID, some_parameters})$$

$$\Delta V_{th_{max}} = f'(\text{time, duty_cycle, toggle_rate, VGS, VDS, ID, some_parameters})$$

We to derive these relations from Cao's model¹², summarized in this table:

TABLE I
SUMMARY OF THE PREDICTIVE MODEL

ΔV _{th} under NBTI			
Static	$A \left((1 + \delta)t_{ox} + \sqrt{C(t - t_0)} \right)^{2n}$		
Dynamic	Stress $\left(K_v(t - t_0)^{0.5} + \sqrt[2n]{\Delta V_{th0}} \right)^{2n}$		
	Recovery $\Delta V_{th0} \left(1 - \frac{2\xi_1 t_e + \sqrt{\xi_2 C(t - t_0)}}{2t_{ox} + \sqrt{Ct}} \right)$		
K_v	$\left(\frac{qt_{ox}}{\epsilon_{ox}} \right)^3 K^2 C_{ox} (V_{gs} - V_{th}) \sqrt{C} \exp\left(\frac{2E_{ox}}{E_o} \right)$		
C	$T_o^{-1} \cdot \exp(-E_a/kT)$		
t_e	$t_{ox} \quad t - t_0 \geq t_1$		
	$t_{ox} \sqrt{\frac{t - t_0}{t_1} - \frac{\sqrt{\xi_2 C(t - t_0)}}{2\xi_1}} \quad \text{otherwise}$		
E_a (eV)	0.49	E_0 (V/nm)	0.335
δ	0.5	K ($s^{-0.25} \cdot C^{-0.5} \cdot nm^{-2}$)	8×10^4
ξ_1	0.9	ξ_2	0.5
T_o	10^{-8}		

15.8.2. EMC table and parameters

NBTI/PBTI in combination with radon doping fluctuations and other sources of intrinsic device variability just interface state fluctuations and contribute in statistical manner to the variation parameters at circuit level ΔVth and Δbeta (see Section 15.1.2 and REALITY deliverable D1.2), hence their table representation follows that one of variability injectors in general.

ΔVth and Δbeta in this approach are assumed to already include the variability

¹² Sarvesh Bhardwaj, Wenping Wang, Rakesh Vattikonda, Yu Cao, Sarma Vrudhula, "Predictive Modeling of the NBTI Effect for Reliable Design", CICC 2006, p. 189



effects of dopant and interface state fluctuations, CD variations, LER variations and layer thickness variations, and more.

In a typical case, for each correlation_geometry, there is a EMC table representing the mentioned parameters, in a correlated fashion, e.g.:

ptoir	delta_vth	delta_beta	r_sd	r_gate
0.00013	+0.00239	-0.0445		
0.00044	-0.00097	-0.0067		
0.000012	-0.00566	+0.0125		
...		

15.8.3. Scaling rules for circuit and use

As statistical parameter, we consider the combined impact of NBTI/PBTI induced interface states with the other sources of intrinsic variations to apply the matched geometry only via variability injectors. Different amount of trap concentrations will capture the effect of the changing stress conditions, e.g., elapsed time, voltage, T etc. For that reason and similar to the time-zero matched geometry, a variability a Pelgrom rule will apply in this case.

15.8.4. Scaling rule for multiple stress conditions

We do not consider multiple stress conditions yet. Instead we consider a corner like analysis approach that assumes all devices of the circuit to be subject to the same stress factors. Hence, all devices hence share the same VMIF compact model chapter. Yet we accommodate for different “flavours” of stressed circuits with all devices subject to the same degradation by accommodating a different VAMIF compact model chapter for each degraded circuit.

15.9. MOSFET hysteresis

Some types of MOSFETs suffer from hysteresis. Such are: SOI with floating body, MOSFETs as used in EEPROM and certain high-k gate MOSFETs. Also NBTI is a kind of hysteresis that is not modeled separately.

The hysteresis is modeled as a voltage source in series with the gate. Modelling details not available

15.10. Variability of MOSFET temporal noise

Temporal noise *as such* is not part of the VAM(IF). It is assumed to be sufficiently covered in the normal design flow. Yet *variability* of MOSFET noise is represented here. Temporal noise sources are:

15.10.1. MOSFET white noise

Also known as (thermal ~, Johnson ~) noise
(No model for distribution)

No report has been found that thermal noise variability is significantly higher beyond the noise level itself, thus not modeled at the moment.

15.10.2. MOSFET 1/f noise

Also known as “flicker noise” (fn)



We assume following model for spectral noise density of a MOSFET's equivalent gate noise

$S_{V_{Geq}} = \frac{KF}{W.L.f}$ where KF is a technology constant, often given explicitly or indirectly in the

SPICE model.

We propose to represent the variability in a dimensionless prefactor, which is 1 on average, and which applies to the gate voltage domain.

$$S_{V_{Geq}} = fn_prefactor^2 \times \frac{KF}{W.L.f}$$

The Gaussian, instantaneous/temporal, distribution of the gate voltage offset itself is separately obtained by integration of S:

$$V_{G_RMS} = fn_prefactor \times \sqrt{\frac{KF}{W.L}} \times \sqrt{\int_{f_{low}}^{f_{high}} \frac{1}{f} \cdot \partial f}$$

$$V_{G_RMS} = fn_prefactor \times \sqrt{\frac{KF}{W.L}} \times \sqrt{\log\left(4 \frac{f_{high}}{f_{low}}\right)}$$

Where f_{high} corresponds to the bandwidth of the circuit node and f_{low} is somewhere between the age of the circuit or the time since turning to accumulation (TBD).

Smaller MOSFETS suffer from ever larger 1/f noise, just as they suffer from ever larger V_{th} variability. Note that the factor W.L (electrical W and L!, as for V_{th}) in the formula represents a Pelgrom rule. This Pelgrom rule (in contrast to the MOSFET V_{th} variability Pelgrom rule) applies to all geometrical correlations.

15.10.3. MOSFET RTS noise

To model RTS, we need:

- (1) The number of states per reference $W*L$ area.
- (2) The distribution¹³ of V_{th} -equivalent amplitudes for a given state for such given $W*L$
 - ⇒ In strong inversion
 - ⇒ In weak inversion¹⁴

The effective V_G equivalent amplitude is inversely proportional with $W*L$, so these amplitudes must be given for a reference W and L . If not given W_{ref} and L_{ref} are assumed 1 μ m.

- (3) To every individual RTS state belongs a set of emission & capture time constants, or described as an overall time constant $\frac{1}{\tau_{RTS}} = \frac{1}{\tau_e} + \frac{1}{\tau_c}$ and a duty cycle. τ_{RTS} and τ_c are

approximately inversely proportional to inversion charge thus drain current density. τ_e is less dependent of the current. The RTS is most effective when τ_e and τ_c are equal, at a current I_{50} .

The distributions of weak inversion amplitudes is very long tailed; the time constant distribution covers multiple decades (Ms to ns).

The detailed description is subtle, yet for the purpose of estimating noise margin, one may suffice with a correlated set of following parameters:

- ⇒ Amplitude in strong inversion

¹³ K. Abe & al, "Analysis of source follower random telegraph signal using nMOS and pMOS array TEG " IEEE IISW 2007, proceedings p. 62

¹⁴ E. Simoen, B. Dierickx, "Critical examination of the relationship between random telegraph signals (RTS) and low-frequency (LF) noise in small-area Si MOSTs", 12th International Conference on Noise in Physical Systems and 1/f Fluctuations, St.Louis, Missouri, USA, 16-20 Aug., 1993



- ⇒ Amplitude in weak inversion (it may be tricky to “invent” a value in case measurements are not available in the other mode. It does not hurt to give as default value zero then))
- ⇒ Reference drain current $I_{50\%}$ at which the duty cycle is 50% (and $\tau_e = \tau_c = 2 \cdot \tau_{RTS}$) for the reference W and L .
- ⇒ The actual time constant τ_{RTS} at $I_{50\%}$.

How to use noise in a digital design flow?
 A possible approach for using this information for the digital design flow would be to adjust Monte-Carlo-wise the noise margin.
 Start from a default noise margin, and add to the margin the amplitude derived from the instantaneous range of eq gate voltages due to $1/f$ and RTS. How to do this in practice and check if this corresponds to reality is an open research topic.

We assume that if both RTS and $1/f$ are described, that they are cumulative. *This is not obvious as RTS and $1/f$ are emanations of the same phenomenon: $1/f$ can be described as the superposition of many smaller RTSs.*

<pre>Object type=mosfet_noise name=<mosfet_type> Value kf Parameter fn_prefactor Parameter rts_states Value w_ref Value l_ref Parameter i_50 Parameter tau_rts Parameter amplitude_si Parameter amplitude_wi</pre>	<p>If name not given, applies to all MOSFETs Optional, rather build on the mechanism present in the existing SPICE models if available Voltage domain prefactor to flicker noise. Average value being 1 number of RTS states in area $W_{ref} \cdot L_{ref}$ Effective (electrical) W Effective (electrical) L $I_{50\%}$, τ_{RTS} and the amplitudes must be a correlated set of values. In case that either $amplitude_{si}$ (strong inversion) or $amplitude_{wi}$ (weak inversion) are lacking, 0 is a good default.</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



16. Other underlying models

16.1. Activity, stress history on circuit parts (cells) and their inputs

Is part of several chapters. E.g. in [chapter cell](#) as it has to be used in the calibration of standard cells and memories for reliability effects.

In order to calibrate cells for activity on their nodes, one needs a representation of “stress history”. This is preliminary and needs huge refinements.

The age of a device is represented in [chapter system](#) as

<pre>Container name=system_properties value age nnn</pre>	<p>in seconds! Si units the number of seconds of calendar lifetime of the device</p>
-------------------------------------------------------------	------------------------------------------------------------------------------------------

The most basic way to represent “activity” information is as:

<pre>Container type=activity [value component name] [value input name] value stress_case name value togglerate nnn value dutycycle nnn</pre>	<p>default cell name: applies to all cells default input: applies to all inputs of cell case: one of: typical, worstcase, standby, ... as a scenario togglerate: the effective number of 0-1 transitions per second dutycycle: the effective ratio between input high time and the total time</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

This approach is certainly too simplistic. One should represent such information per individual cell, and for each cell a number of cases. And discriminations per input, even on power supply tuning or stdby modes. Likely, if too numerous, it will be in a separate file (is there a standard...?)

Another shortcoming is that it does not represent correlation between inputs, not that it represents the recovery time, which is of use in degradation mechanisms that have some recovery or hysteresis.

16.2. Backend definition

In: [Chapter technology](#).

This information is used to define backend variability, including LER, thickness variations...

<pre>Container type=backend_definition //with dielectrics and metals from bottom to top Container Type=dielectric Name=STI Parameter thickness Value permittivity Parameter via_resistance Container Type=interconnect Name=metal1 parameter thickness parameter delta_w // where $W_{physical} = W_{design} + \Delta W$ value min_width //design value</pre>



	<code>value min_spacing</code>	<code>//design value</code>
width!	<code>parameter LER</code>	<code>//NOTE is not the same as distribution on</code>
	<code>[value spacing_permittivity]</code>	<code>//in case the spacing permittivity is different</code>
from lower/upper dielectric	<code>value resistivity</code>	
TBD	<code>value resistivity_offset</code>	<code>// effect of the encapsulation layers etc. formula</code>

16.3. Temperature gradient

temperature gradient as such is not considered as a variability, but a system level input to the simulation flow. In that sense it is a “predicted systematic variability”. External tools might yield sub circuit specific local temperatures based on system operating mode.

How to enter this in the VAM context is not obvious. Possibilities are:

- neglect and work with an average die temperature (given in chapter system)
- back annotated local temperature per subcircuit, eventually percolating down to each transistor. Input should come from external tools, data format unknown.
- clever back annotation, using similar concepts as scenarios for power estimation and historic stress conditions. I propose to piggyback such developments, both for the method to propagate the information, as to the activity/scenario method.



16.4. Standard cell libraries

(Part of [chapter cell](#))

Standard cell libraries are referred to:

Spectre/Spice netlists represent the original variability free library. We also make a clear separation between representation of .lib and of methods to create a .lib

Creation of .lib

in this example STDCELL_ROOT_DIR represents the root directory of the standard cell characterization project (e.g. <file:///einstein/scratch2/marchal/characterization>) and LIB_NAME is a name of a library under characterization (e.g. PT130).

The library may be represented by a collection of .gds files, one for each standard cell. Using an LVS tool (e.g. CALIBRE) this library is converted to a collection of Spectre netlists (the directory STDCELL_ROOT_DIR/run/svbd) .

Representation of .lib

See below

[Standardcell_library](#)

The variability aware library e.g. is produced by SignalStormLC tool (Cadence) which perform characterization of standard cell with applied compact model variability. The output is the .alf library, subsequently converted to the .lib compatible format.

the variable aware .lib library is represented as

- ⇒ For each of the 5 geometries, and as reference one “invariable” library, which is more or less the same as the “typical-typical” library.
- ⇒ one large library file that includes all variable instances of all characterized standard cells (e.g. for the cell INVBD2 cell there are several corresponding variable instances INVBD2_v1, INVBD2_v2 ..., INVBD2_vmax).



Note on cell interpolation:

- Instances of the same cells in .lib files must be interpolatable: i.e. one can make a sensible linear combinations of instances of the same cell, by simply making linear combinations of the numbers on the same position in these cells. *Concept to be proven!*

Geometrical correlation is introduced in the .lib files

<pre>container type=cell_library name=<name of library></pre>	The .lib or vital "library name". If there are multiple libraries used, multiple containers are there.
<pre>Value root <root directory></pre>	If empty, neglect
<pre>Value gds_dir <pathname></pre>	GDS is given in case Spectre or Spice netlist do not exist yet and tool can handle that
<pre>Value spectre_dir <pathname> Value spice_dir <pathname></pre>	Only one is required
<pre>Container Name=Cells temperature =<...> vdd =<...></pre>	Attributes temperature and VDD are modes at which the library is calibrated. There can thus be multiple copies of this container applying to the same library.
<pre>Value dotlib_dir <pathname></pre>	This directory on itself contains 6 subdirectories named invariable, matched, local_systematic, c2c, w2w and b2b. The files therein are the libraries, (one or more files per directory)
<pre>object type=cell name=<cellname> Parameter instantiation Parameter defunct (0 or 1)¹⁵ Parameter outlier (0 or 1)¹⁶</pre>	<i>Instantiation</i> is the name of an instance in the EMC population of cells. Used to retrieve a certain instantiation from the .lib.
<pre>[*(Parameter inputparameters)]</pre>	Optional "Input domain parameters": parameters of underlying blocks (e.g. Vths inside std cells). These values describe the used input parameter. Can be used for binning or RSM if ever one wants to do that. These parameters have geometry obviously corresponding to the

¹⁵ Defunct==1 mandatory parameter, means that this instantiation is a non-functional part. How to handle in the .lib and tools using the .lib remains to be discussed. Minimal approach is to give the non-functional part very large (1 second) delays in the .lib.

¹⁶ Outlier==1 means that the cell contains one or more transistors that were labeled as "outlier" themselves.



	dotlib geometry. Alternate and preferred: use the "corrid ###" concept.
<pre>[Value nominalload [Value inputslope [Parameter nominaldelay¹⁷ [Parameter zeroaddelay [Parameter nominalenergy [Parameter zeroenergy [Parameter staticpower [Parameter nominalsetup [Parameter zeroaddelay [Parameter nominalhold [Parameter zeroaddelay</pre>	<p>These 5 (9 for sequential cells) correlated "output domain" parameters, are optional. They yield help at the next level of propagating variability.</p> <p>Currently implemented using liberty parser and timing engine of Synopsys Design Compiler.</p>

Defunct standard cells
 Due to variability, it is likely that some instances of cells will not yield a correct functionality. Classic calibration software (as SignalStorm) may crash or yield not usable results.
 Yet, such cells must be represented in VAMIF as realistic as possible:

- Such cell instances should have an entry and an instantiation
- They parameter "defunct" is set to 1
- A .lib entry must exist. Minimally it must mimic the actual faulty operation by a very long delay, >>1ms
- Preferably, the .lib entry models the static and dynamic power correctly.
- Normally not possible with .lib, but useful and not required, would be to change the functionality of the cell.

16.5. Representation of non-standard cells: embedded memories

16.5.1. The MemoryVAM configuration contained

<pre>Container type=configuration name=memoryvam</pre>	
<p>Process 1 –related options:</p>	
<pre>list donuts fast_donut l2memory</pre>	<p>This is the list of <i>the</i> donuts to be processed by MemoryVAM. The actual donut information is inside, and referred to by, configuration container <donut_name> donut, within this configuration</p>
<pre>value model.name vdd list model.domain 1.0 1.1 1.2</pre>	<p>Optional. Enumerate modes starting from one, and supply</p>

¹⁷ nominaldelay: the maximum delay for any input to any output, where all input see the given input slope, and all outputs are loaded nominally, eg. the max in the .lib table. zeroaddelay: the maximum delay for any input to any output, for zero load. Energy: dynamic energy, per operation; staticpower: DC [leakage] power.



<pre> value mode1.command ".param vdd_sv=%s" Value mode2.name temperature list mode2.domain 273 300 400 value mode2.command ".temp '%s-273'" Value mode3.name corner list mode3.domain "NN FF SS SF" value mode3.command ".LIB '/path/to/your/lib' %s" Value mode4.name Margin_control_signal list mode4.domain 00 01 10 11 value mode4.command ".inc mcs%s.inc" (...) </pre>	<p>Mode<n>.name, Mode<n>.domain, Mode<n>.command. *.name should not contain spaces.</p>
	<p>NOTE: Some modes you can apply without changing your netlist, such as simple parameter settings (eg: ".param vdd=%s" overrides original value for vdd), while others require you to change the netlist, for example in ".LIB '/path/to/your/lib' %s", while most likely not work if you keep your original .LIB statement. Also NOTE: do not supply multiple .TEMP statements! Remove your original .TEMP statement if you are using .TEMP in a mode or work with ".TEMP placeholder" in the netlist and with ".PARAM placeholder=%s" in the mode command instead.</p>
	<p>Mode names must not contain spaces or special characters! These names will be found back later as attribute names of the donut objects.</p> <p><i>Do not forget that one can always reduce to a trivial configuration space, i.e. the donut is specific for one specific memory, hence these lists have only one value or are simply omitted</i></p> <p>Modes can be overwritten by particular donuts (NOT YET IMPLEMENTED).</p>
<pre> Value n_matched 100 Value n_local_systematic 100 Value n_c2c 100 Value n_w2w 100 </pre>	<p>Number of samples to go for. Specify 0 for any geometry you do not want to simulate.</p>



<pre>Value n_b2b 100</pre>	<p>N_matched also applies to the Islands, see below.</p> <p>Specifying a number >0 for a geometry that is not supplied in the compact model chapter causes an error.</p>
<pre>Value gamma_matched 0 Value gamma_local_systematic 0 Value gamma_c2c 0.9 Value gamma_w2w 1 Value gamma_b2b 0.8</pre>	<p>Gammas (EMC statistics) used for each geometry.</p> <p>You can omit gamma values and MemoryVam tries to find an optimized gamma for you.</p>
<pre>list CORNERS NN FF SS SF ... Value CORNER_LIB '/path/to/your/lib'</pre>	<p>Will create simulation decks using CORNER_LIB with the .LIB suffices given in CORNERS</p> <p>There is no CORNER Geometry Type (yet). Therefore, at the moment, new object in a new cell.xml file in a new directory corners/ is created and the w2w Geometry is used. The user can browse this chapter and combine corner figures and cloud figures using the MATLAB Figure editor functions.</p> <p>(NOTE: STATUS is EXPERIMENTAL. Known limitation: If you use MODES (see above, the behaviour of CORNERS is undefined)</p>
<pre>Value dest_directory <dir></pre>	<p>Working directory for MemoryVam. If omitted, <xml_directory>/.mvam will be used. The directory is created if it does not exist.</p>
<pre>Value pass_to_vaxc_<option> <value> (...)</pre>	<p>Optional low level control of vaccinate. Normally not required for MemoryVAM users. <option>=<value> is processed by vaccinate, not by MemoryVam. Refer to VACCinate user guide for more information and list of supported options.</p>
<pre>Value pass_to_vaxc_gamma_autoreduce 30 Value pass_to_vaxc_gamma_autoreduce_step 0.9</pre>	<p>Automatically reduce too high gamma values to a small enough number using at most gamma_autoreduce iterations and adapting GAMMA -> GAMMA * STEP. If, after all</p>



	<p>iterations, Gamma is still too high, MVAM continues with a warning. DEFAULT: 30 iterations, step factor 0.9. To turn off Gamma autoreduction based on probability, set gamma_autoreduce to 0 (not recommendend). To make the reduction in finer steps, use higher numbers for step, such as 0.95 or 0.99 but then make sure to use higher number of iterations.</p> <p>The default settings are safe together with starting values for gamma of 1.0</p> <p>NOTE: Do not worry about messages like</p> <pre>Warning: Divide by zero. > In pick_matrix>analyze_probabilities at 328 In pick_matrix at 179 In vaccinate at 242 In mvam at 146</pre> <p>These happen when gamma is so high that the resulting probabilities would be smaller than the resolution of a double float. In this case mvam reduces gamma automatically.</p>
<p>Simulator –related options:</p>	
<p>Value simulator_executable <executable></p>	<p>MemoryVAM runs the simulator <executable> for you with the vaccinated donut file name as only parameter from the directory of the vaccinated donut file. You can add command line options here, MemoryVAM always adds the testbench-wrapper as last parameter.</p> <p>Example: “hspice64 –mt 8 –i “ or “hspice_sub –l”</p> <p>NOTE: you can also specify a wrapper script that takes as \$1 parameter the testbench-wrapper to be simulated.</p>
<p>Value unimportant_measures <regex_string></p>	<p>By default, MemoryVAM considers a “failed” of any .MEAS statement as a</p>



	<p>functional failing of the donut variant. You can specify a regular expression for exceptions here. Example: “^chk dummy stability\$” Will ignore “failed” measurements when computing yield for all measures beginning with “chk”, the measure “dummy” and all measure ending with “stability”.</p>
Process 2 –related options:	
<code>Value N 100</code>	<p>Memories are created in a Monte Carlo Fashion from donut statistics. Specify here how many memory samples you want. 100, 200, 5000 are good settings for low, medium, and high accuracy.</p>
<code>List memory_cells M1109 M4563 S6542</code>	<p>Not USED. Every donut produces one memory in this version.</p> <p>The actual memory cells to be processed. The actual cell information is in the container <code>memory_cell</code>, including which donut is used in that cell.</p>



16.5.2. Donuts of memories

Memories as such cannot be calibrated as a whole. In VAM, MemoryVAM calibrates a memory through its “donut”. See also MemoryVAM Users manual.

<code>Container type=donut name=<donut name></code>	This container contains configuration information for one particular donut
<code>Value testbench <path></code>	Test bench = path to file name which contains instantiation of your donut.
<code>Value circuit <circuit name></code>	Needed for VACCination to know where to start injecting. If your circuit resides at toplevel, specify: top_level__ (in total four underscores)
<code>Value island_table <path></code>	Path to the island table. Example: /my/project/island.csv. See below on format of this file. Without island specification, process two cannot run and process one can run with limited functionality only.
<code>Value vsr_table <path></code>	Path to the “Variability scaling rule table”. Example: /my/project/vsr.csv. Without this file, See below on format and meaning of this file.
<code>Value vsr_table <path></code>	Path to the “constraint table”. Example: /my/project/constraints.csv. See below on format of this file.



16.5.3. Memory cells

These are represented at the same level as standard cells and standard cell libraries. Memory cell variability is coded as a population of .lib cells. There may be one or more memory libraries, represented as following container. (see als MemroyVAM container)

<pre>container type=memory name=<name_of_memory></pre>	The .lib or vital "library name". If there are multiple libraries used, multiple such containers exist.
<pre>Value root <root directory></pre>	If empty, neglect
<pre>Value gds_dir <pathname></pre>	Only for reference
<pre>value donut <name></pre>	the donut underlying this memory
<pre>*(object type=memory name=<name> Temperature=60 Vdd=1.1 <configurationparameters>=<...></pre>	All modes and configuration parameters are attributes to the memoryobject
<pre>Value dotlib_dir <pathname></pre>	This directory on itself contains 6 subdirectories named invariable, matched, local_systematic, c2c, w2w and b2b. The files therein are the libraries, one single file per directory.
<pre>Parameter instantiation Parameter defunct (0 or 1) Parameter outlier (0 or 1)]</pre>	Instantiation is the name of an instance in the EMC population of cells. Used to retrieve a certain instantiation from the .lib.
<pre>[*(Parameter corrid_###)]</pre>	See the "corrid_###" concept.
<pre>Parameter <*></pre>	.

16.6. Top level components hierarchy

System yield analysis works with top level components. In the most trivial case there is only one single toplevel component. In the more elaborate case, there may be a hierarchy of toplevel components, which is represented as containers of toplevel components. Toplevel component names correspond with their RTL name and with the names in backannotated netlist container.

<pre>Container type=toplevel_components Value rtl_path <string></pre>	
<pre>Value component <component_name1> Value component <component_name2> container toplevel_component Value component <component_name3> Value component <component_name4></pre>	

Do not confuse this container with the following! The top level description may contain multiple instances of the same component.



16.7. Backannotated netlists of components

System component library, i.e. all cells that are use in yield analysis. Each of the cells is described by VHDL or Verilog, and [may be] backannotated using .SPEF and .SDC file

Custom cells (used in Macro's) and Standard cells do not belong here.

<pre>Container type=netlist name=... Vdd=... Temperature=... Value root <root directory> Value spef_dir <path> Value sdc_dir <path> Value verilog_dir Value vhdl_dir Value sdf_dir Value vcd_dir</pre>	<p>Attributes . The temperature/supply set for which this library is evaluated. There may thus be multiple instances if this container.</p> <p>If empty, neglect Is absent, one neglects backannotation If not given, one uses a default [tbd if VAMIF must provide a default] At least one per cell is needed Optional. Coding of populations of components xxxx_v1 etc. if needed.,. maybe sfd s are not recorded</p>
<pre>Container type=components</pre>	<p>This information allows EMC sampling components For each component there is 1 entry.</p>
<pre>Object Type=component name=<componentname> Parameter longestdelay Parameter staticpower Parameter dynamicenergy Parameter instantiation Parameter defunct (0 or 1) Parameter outlier (0 or 1)</pre>	<p>Instantiation is the name of an instance in the EMC population of components. The longest delay for the actually used set of vectors. Not exactly the same as critical path delay.</p>

The effect of output load on delay and dynamic power is accounted for by the fact that output interconnect C is lumped into the component itself.

16.8. Variability aware yield prediction

The variability aware yield prediction tool consumes information from the digital chapter and produces results written into the system chapter.

The input is represented by three main parts:

- 1) The description of system



- the [possibly hierarchical] list of top level components, implying also the connectivity, which is by default assumed to correspond to a parallel organization.
- 2) The PTOIR versus (dynamic energy, static power, max delay) in EMC format, for each component
 - the component variabilities, from the backannotated netlist
 - 3) The activity of each block (at this moment it's only one number - derived from previous (logic?) simulation - representing the number of activations of the block per unit of time; it's application dependent); future extensions assumes the bit level vector trace (vector of bits) representation [see [activity](#)]
 - 4) The timing and energy constraints enabling yield calculation

The output is

- 1) Multi-dimensional representation of (parametric including functional) variability cloud the user parameter space (clock frequency, power, <other parameters as T, VDD...>)
- 2) This will happen for each geometry separately, and for all geometries aggregated.

Converting to iso-yield curves is done in a rendering tool such as the “VAMIF-browser” (browse). If the rendering tool can do the aggregation itself, no separate parameter is required.

<pre>object type=systemyield Vdd=1.8 Temperature=21 Age=82000000 Value <other> <some value> Parameter totalpower Parameter cycletime Parameter aggregated_totalpower Parameter aggregated_cycletime</pre>	<p>Attributes, multiple instances of the same object may exist</p> <p>All are aggregated in geometry “all (matched)”</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------

16.9. Example of a configuration container

See the MemoryVAM configuration container



17. Specific fast models for Top-Down

The following models are specific for the so-called “Quick&Dirty” VAM. They contain the reduced set representation of libraries and other objects.

At yield estimation level and at compact model abstraction level, one assumes that the scripts are already fast and do not need a Q&D version.

17.1. Q&D standard cell representation

Applies to the “average” Stdcell, or on each individual (if specific name is given TBD).

This information is either extracted as an average form .lib, or synthesized separately and eventually brought back into a .lib by scaling.

Object <code>Type=qd_standardcell</code>	
<code>[Value cellname <name>]</code> <code>Value nominalload</code> <code>Value inputslope</code> <code>Parameter nominaldelay</code> ¹⁸ <code>Parameter zeroaddelay</code> <code>Parameter nominalenergy</code> <code>Parameter zeroenergy</code> <code>Parameter staticpower</code>	Nominalload [F] = load capacitor for which nominal delay and energy are obtained. This q&d model assumes that any other load condition is linearly interpolated Inputslope [V/s] = assumed worst case input slope [default zero]

17.2. Q&D area

In some applications of VAM, it is necessary to have an estimation of the area of a circuit part (standard cell, digital block):

- To estimate the functional yield, using the manufacturing yield object
- To estimate the average distance between circuit part inside, to assess the interpolation of variability between matched and c2c geometries, using correlation length and correlation exponent.

¹⁸ nominaldelay: the maximum delay for any input to any output, where all input see the given inputslope, and all outputs are loaded nominally, eg. the max in the .lib table. zeroaddelay: the maximum delay for any input to any output, for zero load. Energy: dynamic energy, per operation; staticpower: DC [leakage] power.



<code>Container</code> <code>type=qd_area</code>	
<code>Value mosfet_over_node_factor</code>	how much area does a MOSFET (in a stdcell) take on average compared to node number [e.g.22nm] squared
<code>Value stdcell_routing_efficiency</code>	Ratio between actual routed stdcell density and minimum packed stdcells
<code>Value stdcell_over_mosfet</code>	

17.3. Ion ioff

Elementary MOSFET parameters for quick and dirty analog/digital behavior

<code>Container</code> <code>Type=Qd_ion_ioff</code> <code>Name=<Mosfettype></code> <code>Ion (for a minimum MOSFET)</code> <code>IoFF (for a minimum MOSFET)</code> <code>Cgate (for a minimum MOSFET)</code>	
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

17.4. Critical path distribution

Part of chapter digital.

This feature is not used in a full VAM flow, as the detail of each path is known as such and is thus not a variability “parameter”. This parameter is of use in early, fast, empirical path-finding.

[critical] path length is expressed in units of execution time, without further correlations added. [If this distribution is part of a larger set which contains also parameters as VDD and T, we have inherited automatically that correlation.]

<code>object</code> <code>type=critical_path_distribution</code> <code>parameter critical_path</code>	required
-------------------------------------------------------------------------------------------------------------	----------

18. References

References appear in text.

Appendix A

The documentation of VAM IF API implemented in JAVA