Copyright and Trademark Notices

The information contained in this document is subject to change without notice.
Ansoft makes no warranty of any kind with regard to this material, including, but not limited
to, the implied warranties of merchantability and fitness for a particular purpose. Ansoft shall
not be liable for errors contained herein or for incidental or consequential damages in connection
with the furnishing, performance, or use of this material.
This document contains proprietary information that is protected by copyright. All rights are
reserved.
Ansoft Corporation
225 West Station Square Drive
Suite 200
Pittsburgh, PA 15219
(412) 261 - 3200

SIMPLORER, Maxwell2D, RMxprt, and PExpnt are trademarks of Ansoft Corp.
UNIX® is a registered trademark of UNIX Systems Laboratories, Inc.
Windows™ is a trademark of Microsoft Corporation.
FLEXlm is a registered trademark of GLOBEtrotter Inc.
Matlab/Simulink is a registered trademark of Mathworks Inc.
Matlab is a registered trademark of Mathworks, Inc.
Mathcad is a registered trademark of MathSoft Inc.
Adobe Reader is a trademark of Adobe Systems Inc.
FrameMaker is a trademark of Adobe Systems Inc.
Paint Shop Pro is a trademark of Jasc Software Inc.
WinZip is a trademark of Nico Mak Computing, Inc. and Top Systems GmbH.

The examples used in this document were derived from benchmarks developed by the working
group 'VHDL-AMS' of MSR project. The MSR Consortium was founded by German Car
Manufacturers and Suppliers to define and establish standards for information/data exchange
between manufacturer and supplier. MSR has granted permission to use the examples found
in this tutorial. More information on the MSR Consortium and their activities can be found at
The claw-pole design in this document was used with permission from Vahe Caliskan. Vahe
Caliskan disclaims any responsibility or liability resulting from the development and use of ex-
amples in the described manner.

This Manual was created with Adobe FrameMaker.
Some of the pictures were created with Paint Shop Pro.

© Copyright 2003-2004 Ansoft Corporation
Printing History

New editions of this manual include material updated since the previous edition. The manual printing date, indicating the manual’s current edition, changes when a new edition is printed. Minor corrections and updates incorporated at reprint do not cause the date to change.

<table>
<thead>
<tr>
<th>Edition</th>
<th>Date</th>
<th>Software Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>April 2003</td>
<td>6.0</td>
</tr>
<tr>
<td>2</td>
<td>September 2004</td>
<td>7.0</td>
</tr>
</tbody>
</table>
Welcome to the SIMPLORER Simulation Center, the integrated simulator for complex technical systems. Whether you are a beginner or an experienced professional, SIMPLORER’s comprehensive suite of tools gives you expert and reliable results in very little time.

You can create related simulation models quickly and process simulations accurately and reliably with the simulator backplane technology, and you can present and arrange the results with powerful post processors. You can also transfer the simulation data and results to other applications.

This tutorial introduces you to the VHDL-AMS modeling capabilities provided by SIMPLORER.

SIMPLORER is continuously being developed and new functions are added regularly; hence, there may be slight inconsistencies in the documentation.
Welcome
## Table of Contents

1 Introduction 1  
2 VHDL-AMS Models in SIMPLORER 11  
  2.1 Using VHDL-AMS Models 12  
  2.2 Displaying Results 21  
3 Automotive Powernet System Example 25  
  3.1 Alternator 29  
  3.2 Battery 35  
  3.3 DC-DC Converter 40  
  3.4 Ignition Switch and Loads 45  
  3.5 Configuration Modes 50  
  3.6 Powertrain 56  
4 Case Studies 63  
  4.1 Different Modeling Styles: Battery 65  
  4.2 Automated Model Development Using VHDL-AMS Wizard: Fuse 71  
  4.3 Detailed and Average Model Development: Claw-Pole Alternator 81  
  4.4 Multilevel Modeling Techniques: Loads 90  
  4.5 Multidomain System Modeling: Linear Drive System, Solenoid 103  
  4.5.1 Linear Drive System 103  
  4.5.2 Solenoid System 108  
  4.6 Mixed-Signal Modeling: DC-DC Model with PWM, Automotive Alarm System 114  
  4.6.1 DC-DC Model 114  
  4.6.2 Automotive Alarm System 118  
5 Model Development 131  
  5.1 VHDL-AMS Subsheets in Schematic 132  
  5.2 Library and Model Management in the ModelAgent 135  
  5.3 Managing VHDL-AMS Models in the ModelAgent 141  
  5.4 Managing VHDL-AMS Packages in the ModelAgent 158  
  5.5 Simulator Environment 162  
6 VHDL-AMS Language Fundamentals 165  
  6.1 Design Units 166  
  6.1.1 Entities and Architectures 166  
  6.1.2 Packages 168  
  6.2 Subprograms 172  
  6.3 Declarations 174  
  6.3.1 TYPE Declarations 174  
  6.3.2 Data Object Declarations 175
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.3</td>
<td>Other Declarations</td>
<td>178</td>
</tr>
<tr>
<td>6.4</td>
<td>Concurrent Statements</td>
<td>179</td>
</tr>
<tr>
<td>6.5</td>
<td>Sequential Statements</td>
<td>184</td>
</tr>
<tr>
<td>6.6</td>
<td>Simultaneous Statements</td>
<td>191</td>
</tr>
<tr>
<td>6.7</td>
<td>Identifiers, Literals, and Expressions</td>
<td>195</td>
</tr>
<tr>
<td>6.8</td>
<td>Predefined Data Types</td>
<td>198</td>
</tr>
<tr>
<td>6.9</td>
<td>Predefined Attributes</td>
<td>199</td>
</tr>
<tr>
<td>6.10</td>
<td>Reserved Words</td>
<td>201</td>
</tr>
<tr>
<td>6.11</td>
<td>Modeling Aspects in SIMPLORER</td>
<td>205</td>
</tr>
<tr>
<td>7</td>
<td>Appendix</td>
<td>211</td>
</tr>
<tr>
<td>7.1</td>
<td>SIMPLORER Glossary</td>
<td>211</td>
</tr>
<tr>
<td>7.2</td>
<td>Table of SIMPLORER Libraries</td>
<td>217</td>
</tr>
<tr>
<td>7.3</td>
<td>Common Conventions in SML</td>
<td>218</td>
</tr>
<tr>
<td>7.4</td>
<td>Troubleshooting</td>
<td>225</td>
</tr>
<tr>
<td>7.5</td>
<td>Literature Reference</td>
<td>226</td>
</tr>
<tr>
<td>8</td>
<td>Index</td>
<td>227</td>
</tr>
</tbody>
</table>
General Description of Content

1 Introduction
   Describes the general procedure for solving a simulation problem, lists general hints for using
   this manual, and explains how to install SIMPLORER.

2 VHDL-AMS and VHDL Models in SIMPLORER
   Describes the modeling of VHDL-AMS models using the graphical input application Schematic,
   and introduces the use of subsheets and macros.

3 Automotive Powernet System Example
   Describes the modeling and use of VHDL-AMS models in a system-level automotive powernet
   example. This example is developed in six steps using simplified models for powertrain, battery,
   alternator, DC-DC converter, and load models in VHDL-AMS. It also illustrates the use
   of configurations.

4 Case Studies
   Uses six case study examples to provide an in-depth understanding of how to use the VHDL-
   AMS language for different modeling tasks. These examples describe the use of multiple model
   architectures for different levels of complexity, different modeling styles, and different behavior.
   Examples that illustrate the multidomain and digital modeling flexibility of VHDL-AMS are also included.

5 Model Development
   Describes how to create model libraries, models, and packages in the ModelAgent, an applica-
   tion used to manage all libraries and models in SIMPLORER.

6 VHDL-AMS Short Reference/Language Fundamentals
   Describes the VHDL-AMS language syntax.

7 Appendix
   Provides a glossary of frequently used terms and a literature reference list for SIMPLORER.

8 Index
   Offers a comprehensive listing of keywords and associated information used in the VHDL-
   AMS Tutorial.
1 Introduction

SIMPLORER is a software package used to design and analyze complex technical systems. Simulation models created with SIMPLORER can contain circuit components from different physical domains, block elements, and state machine structures modeled in SML as well as VHDL-AMS.

SIMPLORER’s simple graphical interface makes even complex models easy to design. Fast and stable simulation algorithms reduce simulation time and provide reliable results.

The various tools used for modeling, simulating, and analyzing are integrated within the SIMPLORER Simulation Center (SSC). The SSC Commander starts the programs, manages the project files, and sets options for both simulation and program environment.

SIMPLORER performs calculations for simulation models described in VHDL-AMS. VHDL-AMS stands for Very high-speed integrated circuit Hardware Description Language – Analog Mixed Signal. The SML compiler automatically starts the VHDL-AMS simulator if VHDL-AMS components are used in the simulation model. Standard components for VHDL-AMS simulation are available on the «AMS» tab, the «Digital» tab, and the «Tools» tab of the ModelAgent.

Content of the Tutorial

This Tutorial describes the special functionality for modeling elements in VHDL-AMS within the SIMPLORER environment. This includes an introduction to the use of VHDL-AMS components as well as general model development in SIMPLORER. The simulation models used in the examples in this manual are included on the CD that accompanies the SIMPLORER student version. The examples can be loaded into SIMPLORER from the included files or can be created by following the step-by-step instructions in this Tutorial.

The Tutorial does not teach engineering design or cover the complete VHDL-AMS language syntax. These topics are large enough by themselves to warrant several books.

Tutorial Content on the Student Version CD

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Autorun files</td>
</tr>
<tr>
<td>Documentation</td>
<td>SIMPLORER documentations with installation guide and Getting Started Guide in PDF format</td>
</tr>
<tr>
<td>Examples</td>
<td>Examples used in the Tutorial including library and project files</td>
</tr>
<tr>
<td>SIMPLORER7</td>
<td>SIMPLORER program files, model libraries, and examples</td>
</tr>
<tr>
<td>Tools</td>
<td>Adobe Reader</td>
</tr>
<tr>
<td></td>
<td>HTML help compiler and support files</td>
</tr>
<tr>
<td></td>
<td>DAO (Data Access Object) setup files</td>
</tr>
<tr>
<td></td>
<td>WebUpdate setup files</td>
</tr>
<tr>
<td></td>
<td>PowerPoint Viewer</td>
</tr>
</tbody>
</table>
SIMPLORER is fully Windows compliant (Win2000/XP). The basic operation of the windows and user environment is consistent with the Windows standard, and therefore is not described here. For questions about Microsoft Windows, please consult Windows documentation. The following formatting conventions are used in this documentation:

<table>
<thead>
<tr>
<th>&lt;Apply&gt;</th>
<th>Button to confirm selection activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE &gt; OPEN</td>
<td>Menu sequence to start an action</td>
</tr>
<tr>
<td>«Properties»</td>
<td>Text on menus and option fields</td>
</tr>
<tr>
<td>QUANTITY</td>
<td>VHDL-AMS keyword</td>
</tr>
<tr>
<td>v == v * t;</td>
<td>VHDL-AMS modeling description</td>
</tr>
<tr>
<td>☑</td>
<td>Selected check box in a dialog</td>
</tr>
<tr>
<td>☐</td>
<td>Cleared check box in a dialog</td>
</tr>
<tr>
<td>☸</td>
<td>Selected option button in a dialog</td>
</tr>
<tr>
<td>☹</td>
<td>Deactivated option button in a dialog</td>
</tr>
</tbody>
</table>

Indicates a helpful hint for using a feature

Indicates important information – Please note!

Content signpost
Indicates detailed specifications of content at the beginning of chapters and sections

Function sequences
Indicates the beginning of instructions that describe the procedure for a complex task

In many cases, clicking the right mouse button, or right-clicking, on an object displays a shortcut menu providing quick access to object activities or properties.

The terms click or mouse click used in this Tutorial refer to the left mouse button.
SIMPLORER Documentation

Use the following guides and manuals to quickly find help while working with SIMPLORER.

Printed Manuals

- **Getting Started**
  Program functionality at a glance, new functions, step by step simulation examples, program conventions.

- **VHDL-AMS Tutorial**
  Detailed descriptions and examples of SIMPLORER's VHDL-AMS functionality (this Manual).

- **Installation Guide**
  Descriptions of how to install Ansoft software products including SIMPLORER.

Online Help

- **Online Help**
  Online Help with index and search capabilities is available for all SIMPLORER programs and installed models.
  Online Help also has a description of additional SIMPLORER modules such as Interface and Coupling elements, C-interface, and optimization algorithms.

- **Examples**
  Examples are available for installed models. Open the example by right-clicking on the model in the ModelAgent and selecting «Example». If SIMPLORER is installed in the default location, application examples are available under C:\Ansoft\Simplorer70\Examples\Applications

Installing SIMPLORER

**Hardware and Software Requirements**

- **CPU:** Pentium 800 MHz or better
- **Memory:** 512 MB or more
- **Free Hard disk:** 2 GB or more
- **Operating system:** Windows 2000/XP
- **Graphic card:** VGA 1024x768

The complete installation needs approximately 350 MB of hard disk space; however, the output of simulation data may require much more hard disk space.

For more information, see the Ansoft PC Installation Guide.

**Installation under Windows 2000 and Windows XP**

1. Save the license.txt file emailed to you by Ansoft as an attachment (for purchased software only).
2. Install the hardware key, if it is necessary. Depending on the type of key you have, it may connect to the parallel printer port or a USB port of the computer.
3. Log in as Administrator for Windows 2000/XP systems. (Otherwise, a message box from InstallShield will appear reporting error -115 when files are being copied.)
Run the FLEXlm setup (once for the whole network, usually on the server or, on the single machine if a single machine version is installed).

Run the SIMPLORER setup.

Starting Setup

1. Put the SIMPLORER CD-ROM into the CD-ROM drive. Setup starts automatically.
2. To start the installation, click <Install Software>.
3. Click <Install SIMPLORER>.
4. Follow the instructions in the setup program.
   If the setup program does not start automatically, use Windows Explorer to open the Autorun.exe file on the SIMPLORER installation disk.

SIMPLORER Version Size Limits

The following table shows the simulation model limits in the SIMPLORER Student Version. Most of the examples used in the Tutorial can be simulated with this version. If there are any restrictions, a text comment explaining the restriction is provided on the sheet in the included example file.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
<th>Student Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conservative nodes (nets)</td>
<td>Electrical circuit nodes (nets)</td>
<td>15</td>
</tr>
<tr>
<td>Circuit</td>
<td>Basic circuit components</td>
<td>30</td>
</tr>
<tr>
<td>Block diagram</td>
<td>Basic block components</td>
<td>15</td>
</tr>
<tr>
<td>State graph</td>
<td>Transfer conditions</td>
<td>30</td>
</tr>
<tr>
<td>VHDL-AMS</td>
<td>Quantities</td>
<td>50</td>
</tr>
<tr>
<td>VHDL-AMS</td>
<td>Signals</td>
<td>100</td>
</tr>
<tr>
<td>C-interface</td>
<td>DLLs to integrate models</td>
<td>0</td>
</tr>
<tr>
<td>Graphical Subsheets</td>
<td>Embedded substructures in Schematic</td>
<td>1</td>
</tr>
<tr>
<td>Text macros</td>
<td>Used text macros</td>
<td>no limit</td>
</tr>
<tr>
<td>Display Elements</td>
<td>Elements for graphical output on sheet</td>
<td>4</td>
</tr>
<tr>
<td>Drawing elements</td>
<td>Drawing elements in Schematic</td>
<td>no limit</td>
</tr>
</tbody>
</table>
Starting the Simulation System SIMPLORER

1. SIMPLORER can be started like all Windows applications using the Start menu on the Windows task bar. Click <Start>, and select the following entries: Programs> Ansoft> SIMPLORER 7.0> SIMPLORER Simulation Center 7.0. The SIMPLORER start-up screen appears.

2. Confirm the displayed user name which was created during setup.

   To add a user name, enter the new user name in the name box. A special profile (directories, screen layout) is associated with each user name in the SIMPLORER program environment.
Introduction

Choose <OK> to start SIMPLORER. The SSC Commander (SIMPLORER Simulation Center) and the Welcome screen appear. The Welcome screen provides four ways to start working:

- Create a New Project.
- Create a New Simulation Model.
- Open an Existing Project.
- Open an Existing Simulation Model.

Click on the button «Create a New Simulation Model» to start the Schematic.

A project can be created before starting Schematic. If a project is created, all the files used for the simulation task in that project are linked to the project and available in the SSC window. Any of the files can be edited or started from the open project in the SSC Commander.

To use the example files provided for this Tutorial, click on the button «Open an Existing Project», or choose PROJECT > OPEN from the open SCC window and navigate to the system example (system_examples.ssc) or case study example (case_study_examples.ssc) files in the Tutorial Examples folder.

Creating a New Project and Starting SIMPLORER Schematic

The SSC Commander is the central communication module of SIMPLORER. The applications are started, projects are managed, and program environments options are defined here.
Creating a New Project

1. Define a name for the project file (project.scc).

2. If desired, enter a project title; otherwise, leave this field blank.

3. Click on <Create> to define the new project.

After a new project is created the SSC window appears, displaying the application list (the window below shows the files in an existing project).

Only one project can be open at any time in the SSC window.

After opening an existing project, click <Schematics> in the application list and double-click on a sheet to open it.
Open multiple sheets by selecting the sheets in the SCC window and double-click «Schematic» in the application list to open them.

Schematic opens automatically with an empty sheet, and displays a Tip of the Day window. To prevent the Tip of the Day window from appearing when Schematic starts, clear «Show tips at startup» before closing the window.

The screen layout may be different from what is shown in the picture above, depending on the View menu settings.
Schematic Window

In SIMPLORER Schematic, simulation models can be created, simulations can be executed, and simulation results can be displayed. A variety of drawing elements can be used to illustrate the simulation model.

The Schematic main window consists of the following elements, accessed by tabs, which can be shown and hidden with the View menu commands.

- **ModelAgent**: Shows the installed libraries and components
- **Object Browser**: Lists the elements available on the Schematic and their properties
- **Build Window**: Displays messages from the Schematic
- **Simulation Script Window**: Displays the script that was used for simulating the design in the Schematic or for exporting a VHDL-AMS description from the Schematic
- **Simulator Window**: Displays messages from the Simulator
- **Compiler Window**: Displays messages from the Compiler
- **Editor Window**: Displays the compiler messages that were generated by compiling a text model on sheet with the embedded Editor
- **Report Browser**: Lists all components of a simulation model and their parameters
- **Analysis Script Window**: Displays the Visual Basic script used for executing an advanced analysis
- **Analysis Log Window**: Displays a log of operations after the completion of an advanced analysis
Ansoft recommends downloading the latest SIMPLORER Product Update to ensure optimal operation of SIMPLORER.

Choosing HELP > WEBUPDATE in the SSC Commander, lists all new SIMPLORER modules available from Ansoft's servers. If an update is available, check the box next to the application in the list. Click <Next> to start the WebUpdate and follow the instructions on the screen. Close all open SIMPLORER programs to avoid possible loss of data. After carrying out WebUpdate, the most recent version of SIMPLORER opens.
2 VHDL-AMS Models in SIMPLORER

VHDL-AMS (Very high-speed integrated circuit Hardware Description Language – Analog Mixed Signal) is a standardized language used for describing digital, analog, and mixed-signal systems.

The Institute of Electrical and Electronics Engineers (IEEE) standardized the VHDL-1076 language as a Hardware Description Language (HDL) for digital models. The VHDL standard from 1993 was extended in 1999 for the description of analog and mixed-signal models in the form of the IEEE 1076.1 standard for VHDL-AMS.


<table>
<thead>
<tr>
<th>Description and simulation of event driven systems</th>
<th>Description and simulation of mixed-signal circuits and systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE 1076 VHDL</td>
<td>IEEE 1076.1 VHDL-AMS</td>
</tr>
<tr>
<td>1993</td>
<td>1999</td>
</tr>
</tbody>
</table>


SIMPLORER supports the development and simulation of models that have been developed in VHDL-AMS:

- VHDL-AMS Wizard can easily and quickly create simple and complex models without the hassle of typing code and correcting syntax.
- VHDL-AMS models in ASCII text can easily be imported into a library or a subsheet on the Schematic.
- VHDL-AMS models in libraries as well as text or graphical subsheets can be easily exported to ASCII text files.
- Schematics containing VHDL-AMS models can be exported as netlists to VHDL-AMS ASCII files.
- Stimulus generator can be used to create digital stimulus in VHDL and supports a variety of patterns and data types.
- VHDL-AMS models can instantiate SIMPLORER models as foreign models - this helpful feature allows users to take advantage of the large number and variety of highly optimized and fast SIMPLORER models.

This chapter contains information on:

- VHDL-AMS models
- Load reference arrow system of physical domains
- Packages and libraries
- Entities and architectures
- Schematic environment
- Model output definitions
- 2D Digital Graph
2.1 Using VHDL-AMS Models

SIMPLORER provides several libraries of VHDL-AMS components developed according to IEEE 1076.1 (VHDL Analog and Mixed Signal Extensions Standard) and IEEE 1076 (VHDL standard):

- Basic VHDL-AMS library (basic_vhdlams.smd located on the «AMS» tab): contains common basic circuit components and blocks
- Digital Elements library (digital_elements.smd located on the «Digital» tab): contains common basic components used for simple digital circuits
- Tools library (transformations.smd located on the «Tools» tab): contains auxiliary components (called OmniCasters) that easily connect different data types and natures

The models provided are open and can be used to derive more advanced models by copying the description to a user library and editing the text to modify the model. The files can be used and distributed if the copyright statement, included in each model description, is not removed. To access the model description, double-click the component name in the Model Agent and select the «Model Text» tab.

The functionality of all VHDL-AMS models is a subset of that of the equivalent SIMPLORER models available in the «Basics» tab. VHDL-AMS models can be used in parallel with SIMPLORER models.

The digital models operate with digital signals and can be characterized with rise time/fall time/propagation delays. They do not have any conservative nodes but can be connected with analog quantities using OmniCasters (in the Tools library).

Across and Through Quantities of Natures

VHDL-AMS models can support nature types for several physical domains. Nature types are properties of conservative nodes (also referred to as ports or terminals) of models. At least one specific nature exists for each domain. An across and a through quantity is associated with each nature. The following table links the across and through quantities for each nature type:

<table>
<thead>
<tr>
<th>Nature</th>
<th>Across</th>
<th>Through</th>
<th>Circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELECTRICAL</td>
<td>Voltage [V]</td>
<td>Current [A]</td>
<td>T1</td>
</tr>
<tr>
<td>FLUIDIC</td>
<td>Pressure [Pa]</td>
<td>Flow Rate [m³/s]</td>
<td></td>
</tr>
<tr>
<td>MAGNETIC</td>
<td>Magneto Motive Force [A]</td>
<td>Magnetic Flux [Vs]</td>
<td></td>
</tr>
<tr>
<td>TRANSLATIONAL</td>
<td>Displacement [m]</td>
<td>Force [N]</td>
<td></td>
</tr>
<tr>
<td>TRANSLATIONAL_V</td>
<td>Velocity [m/s]</td>
<td>Force [N]</td>
<td></td>
</tr>
<tr>
<td>ROTATIONAL</td>
<td>Angle [rad]</td>
<td>Torque [Nm]</td>
<td>T2</td>
</tr>
<tr>
<td>ROTATIONAL_V</td>
<td>Angular Velocity [rad/s]</td>
<td>Torque [Nm]</td>
<td></td>
</tr>
</tbody>
</table>

Load Reference Arrow System

The following table shows the load reference arrow system in SIMPLORER for all available domains. This system defines how across and through quantities are measured for models in each domain. The measuring direction is marked by the red dot on the model symbol. The red dot is always at pin 1 of a model.
• Across quantities: Value is calculated by subtracting the value at Pin2 from the value at Pin1
• Through quantities: Value is positive if the quantity flows into the model at the pin marked with the red dot

<table>
<thead>
<tr>
<th>Difference Sources</th>
<th>Flow Sources</th>
<th>Passive Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fluidic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magnetic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mechanic Translational</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mechanic Rotational</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thermal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Electrical
  - Across quantities: \( V_{i} \) at Pin1, \( V_{i} \) at Pin2
  - Through quantities: \( \Delta P \)

- Fluidic
  - Across quantities: \( Q \)
  - Through quantities: \( \Delta P \)

- Magnetic
  - Across quantities: MMF, FLUX
  - Through quantities: MMF, FLUX

- Mechanic Translational
  - Across quantities: F
  - Through quantities: F

- Mechanic Rotational
  - Across quantities: \( \Phi \), \( \omega \)
  - Through quantities: \( \Phi \), \( \omega \)

- Thermal
  - Across quantities: \( H \)
  - Through quantities: \( H \)
SIMPLORER model libraries (.smd files) can also serve as VHDL-AMS model libraries and house packages and models.

The following symbol notations are used to distinguish between the different types of models and packages within a SIMPLORER model library.

- **Collection of packages**
- **Individual package**
- **Collection of models**
- **Macro model developed either as a text model in VHDL-AMS/SML or imported graphical subsheet**
- **C-model defined in a separate DLL**
- **Internally implemented model**

Packages are collections of re-usable declarations and definitions such as types, constants, functions, procedures, and natures. Packages appear under green package folder symbols in the tree and have a green rectangle in front of their names. Standardized packages from IEEE (such as `math_real` and `textio`) and proposed packages from IEEE (such as `electrical_systems` and `thermal_systems`) are available in the `ieee` and `std` libraries on the «AMS» tab.

Model and package symbols that have a blue padlock on them indicate that the models or packages are locked and cannot be edited.

To use declarations from a package, the corresponding package must be included in the model description, and the package must be available in an installed model library within the current SIMPLORER configuration. In the following example the Resistor model uses the ELECTRICAL nature for its conservative pins. Consequently, the ELECTRICAL_SYSTEMS
User-defined packages can be added to a model library and used in model descriptions. To create a new package within a library, click on the required library and then choose NEW>PACKAGE in the ModelAgent's shortcut menu.

To use model libraries with names that include spaces, an alias must be defined for the name. Choose OPTIONS>PROGRAM DIRECTORIES in the SSC Commander and select the «Aliases» tab to view current aliases and define new aliases for model library names. See “Alias for File Names” on page 209 for more details.
VHDL-AMS Models in SIMPLORER

16 VHDL-AMS Models in SIMPLORER

Entities and Architectures of VHDL-AMS Models

VHDL-AMS models consist of two parts: an **entity** declaration and one or more **architecture** descriptions. The entity describes the interface of the model and declares inputs, outputs, constant value parameters, conservative pins, etc. The architecture defines the behavior of the model, and several modeling styles may be used for this description such as behavioral, dataflow, structural, etc. It is possible to associate multiple architectures with an entity declaration, and only the selected architecture will be used during simulation.

The model descriptions for two of the capacitor models (a plain capacitor and a nonlinear capacitor) included with the system examples for this Tutorial follow. Explanations of the statements used in the model description are included. The system examples project must have been opened as described on page 6 to be able to view these models as instructed.

**Capacitor**

This model has an interface with two electrical terminals and two non-conservative inputs for the static capacitance value and initial voltage value. To view the model description, double-click the capacitor on the «Projects» tab of the ModelAgent, and click the «Model Text» tab.

```vhdl
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY cap IS
  GENERIC (c_nom: CAPACITANCE := 1.0;
            v_init: VOLTAGE := 0.0);
  PORT (TERMINAL p,m : ELECTRICAL);
END ENTITY cap;

ARCHITECTURE behav OF cap IS
  QUANTITY v ACROSS i THROUGH p TO m;
BEGIN
  BREAK v => v_init;
  i == c_nom*v'DOT;
END ARCHITECTURE behav;
```

**Capacitor**

This model has an interface with two electrical terminals and two non-conservative inputs for the static capacitance value and initial voltage value. To view the model description, double-click the capacitor on the «Projects» tab of the ModelAgent, and click the «Model Text» tab.

```vhdl
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY cap IS
  GENERIC (c_nom: CAPACITANCE := 1.0;
            v_init: VOLTAGE := 0.0);
  PORT (TERMINAL p,m : ELECTRICAL);
END ENTITY cap;

ARCHITECTURE behav OF cap IS
  QUANTITY v ACROSS i THROUGH p TO m;
BEGIN
  BREAK v => v_init;
  i == c_nom*v'DOT;
END ARCHITECTURE behav;
```

**Capacitor**

This model has an interface with two electrical terminals and two non-conservative inputs for the static capacitance value and initial voltage value. To view the model description, double-click the capacitor on the «Projects» tab of the ModelAgent, and click the «Model Text» tab.

```vhdl
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY cap IS
  GENERIC (c_nom: CAPACITANCE := 1.0;
            v_init: VOLTAGE := 0.0);
  PORT (TERMINAL p,m : ELECTRICAL);
END ENTITY cap;

ARCHITECTURE behav OF cap IS
  QUANTITY v ACROSS i THROUGH p TO m;
BEGIN
  BREAK v => v_init;
  i == c_nom*v'DOT;
END ARCHITECTURE behav;
```

**Capacitor**

This model has an interface with two electrical terminals and two non-conservative inputs for the static capacitance value and initial voltage value. To view the model description, double-click the capacitor on the «Projects» tab of the ModelAgent, and click the «Model Text» tab.

```vhdl
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;

ENTITY cap IS
  GENERIC (c_nom: CAPACITANCE := 1.0;
            v_init: VOLTAGE := 0.0);
  PORT (TERMINAL p,m : ELECTRICAL);
END ENTITY cap;

ARCHITECTURE behav OF cap IS
  QUANTITY v ACROSS i THROUGH p TO m;
BEGIN
  BREAK v => v_init;
  i == c_nom*v'DOT;
END ARCHITECTURE behav;
```
Nonlinear Capacitor

This model has an interface with two electrical terminals and one non-conservative input for the variable capacitance value. To view the model description, double-click the component on the «Projects» tab of the ModelAgent, and click the «Model Text» tab.

**LIBRARY** IEEE;
**USE** IEEE.ELECTRICAL_SYSTEMS.ALL;

**ENTITY** NonlinearCapacitor IS
**PORT** ( TERMINAL p,m : ELECTRICAL);
**QUANTITY** capacitance : CAPACITANCE := 1.0e-6;
**END ENTITY** NonlinearCapacitor;

**ARCHITECTURE** behav OF NonlinearCapacitor IS
**QUANTITY** voltage ACROSS current THROUGH p TO m;
**QUANTITY** charge : CHARGE := capacitance*voltage;
**BEGIN**
charge == capacitance * voltage;
current == charge'DOT;
**END** behav;

**ENTITY** Interface description of the model NonlinearCapacitor.

**PORT** Conservative and non-conservative pins of a circuit.
Here, there are two electrical terminals that represent the plus (+) and minus (−) pins of the model and one non-conservative input for the capacitance value of the model.

**QUANTITY** Continuous time-varying analog value represented as a non-conservative pin.
Here, capacitance is a variable non-linear capacitance value that is of mode IN (input) to the model. Other common modes are OUT and INOUT (for bidirectional pins).
The data type associated with the capacitance is of type REAL, and the default value is 1 micro farad.

**TERMINAL** Conservative pin associated with a domain.
Here, there are two conservative pins p and m declared for the electrical domain. Other common domains that are used include TRANSLATIONAL, ROTATIONAL, THERMAL, FLUIDIC, and so on.

**ARCHITECTURE** Defines the behavior behav of the model NonlinearCapacitor.
**QUANTITY** voltage ACROSS current THROUGH p TO m;
This statement defines voltage as an across quantity and current as a through quantity between the pins p and m for the model.
current == capacitance * voltage'DOT;
This statement specifies the model behavior in equation form. 'DOT' is the pre-defined attribute to differentiate a quantity (in this case, voltage) with respect to time.
Placing and Connecting Models

The ModelAgent embedded in the Schematic provides VHDL-AMS models on the «AMS», «Digital», and «Tools» tab. To place a component, select a tab and then a component in the tree. Drag the component onto the sheet (drag-and-drop). See also “Schematic Window” on page 9.

The search feature (available only in Query mode) minimizes the complexity of the search and allows quick access to all models. The shortcut command «Query Mode» can be used to change the mode.

Wire mode allows connections to be made between a number of components. To enter wire mode, choose CONNECT > WIRE, or type CTRL+W. Wire mode can also be activated by placing the cursor over a connection point (it changes to the wire cursor) and clicking. Place the cursor over a second point and click to make a connection to that point. Continue to click on the beginning and end points to make connections. Clicking while not on a connection point allows corners to be set to change the direction of the wire. To exit wire mode, press ESC.

Components can also be connected by overlapping their pins, or can be connected to a wire by placing a component’s pin over the wire. Unconnected wires are shown as broken red lines.

Using Transformation Models

Complex simulation models usually need simple auxiliary models to connect different data types and natures simply and quickly, so that the real subject of simulation can be investigated easily. VHDL-AMS models need transformation models that perform data type conversions, and VHDL-AMS models from different domains need transformation models that connect conservative nodes of different natures.

OmniCasters are interface models that are used to interface analog quantities with digital signals or connect digital signals of different data types.

The flexible OmniCaster model, which performs conversions depending on the connected data types, and the fixed OmniCaster models with predefined data types are available for use. The flexible OmniCaster model will automatically use a fixed OmniCaster model based on the data types of the pins connected to it. The fixed OmniCaster models have an ENTITY description according to the nature of the conversion and an ARCHITECTURE description that may use function calls to perform the conversion. The different data types considered for signals are REAL, INTEGER, BIT, BOOLEAN, BIT_VECTOR, STD_LOGIC, and STD_LOGIC_VECTOR. The only data type considered for analog quantities is REAL. Some transformations can be specified with propagation delay, rise time, fall time, threshold, and output value parameters. The functions used for the type conversions are available in the omnicastern_package.

Conservative nodes of different domains can be connected using a Domain-to-Domain (D2D) model available in the Nature Transformations folder of the transformations.smd library. It is also possible to connect a conservative node to a non-conservative node using a C2NC connection model. In this case, the across value from the conservative node is transferred to the non-conservative node. Unlike the OmniCasters that are described in VHDL-AMS, the D2D and C2NC are not VHDL-AMS models. Consequently, if schematics that use D2D or C2NC models are exported to an ASCII netlist, the exported description may not simulate in a third party VHDL-AMS simulator.

See also Chapter 3.6 Step 6: “Powertrain” on page 56.
Defining Model Properties

Every VHDL-AMS model placed on the sheet has a Properties dialog where its parameters can be modified, simulation languages and architectures can be selected. Unlike SIMPLORER models, VHDL-AMS models do not use component dialogs for parameterization of the components. All parameters are simply listed and changed in the «Parameters» tab.

To open the Properties dialog, do one of the following:
- Double-click the component
- Right-click on the component, and choose «Properties»
- Choose EDIT>PROPERTIES for the selected component

The number of parameters in the «Parameters» and «Output/Display» dialog varies depending on the selected model. To enter a parameter value, click in the input field of the corresponding parameter and enter a numerical value (with or without suffix), a variable, or an expression. Default values are used if no other value is defined for the parameter.

The «Library» tab shows information about the model's source library and the model type identification. These settings cannot be changed.

In addition, if the model is described in more than one language, such as VHDL-AMS and SML, then either one of the two descriptions can be selected for simulation. If a VHDL-AMS model has multiple architectures, then any one of the architectures can be selected for simulation. See also “Creating Models in VHDL-AMS in the ModelAgent” on page 142 and Chapter A.3 “Common Conventions in SML” on page 218.

Using Parameter Names

All model parameters can be used in expressions defining other model parameters and on the right-hand side of equations.

Values of model parameters can be accessed using the following: Name.Qualifier, where Name refers to the name of the model and Qualifier refers to the name of the model parameter. For example, to access the resistance value \( r \) of the resistor model \( \text{resistor1} \), use \( \text{resistor1}.r \). Look at the «Output/Display» tab, where all model parameters are listed with their qualifiers, to find the qualifier needed.

In SIMPLORER, it is possible to connect components using the Name.Qualifier syntax. For example, the rotor speed \( n \) of an induction machine \( \text{im1} \) can be connected to the input of a gain block (input) by entering \( \text{im1}.n \) as the «Value» in the gain block's input parameter. While
such connections can be made between VHDL-AMS models within the SIMPLORER environment for simulation, it is not possible to generate and export the VHDL-AMS descriptions of schematics containing these types of connections. Therefore, if schematics need to be exported to ASCII files, then connections between components should be made explicitly with wires.

Parameter names used in SIMPLORER are case sensitive while VHDL-AMS models are inherently case insensitive. All model names and parameter names for VHDL-AMS models in SIMPLORER use lowercase.

See also Chapter A.3 “Common Conventions in SML” on page 218.
2.2 Displaying Results

VHDL-AMS model inputs, outputs, locally defined elements, and values within submodels can be viewed using appropriate analog or digital Display Element.

SIMPLOER allows users to monitor the locally defined values by viewing them in display graphs, and also allows the use of these values as variables in other models. However, this is strongly discouraged if the Schematic has to be exported as a VHDL-AMS description.

Selecting Model Outputs

Display Elements found on the «Displays» tab of the ModelAgent allow the viewing of simulation outputs during and after the simulation. Simulation quantities can be selected and their representations defined from the Properties dialog of the Display Element. The format characteristics of the representation can be set up before or after a simulation is run.

A probe is a quick way of selecting a parameter for display. The probe parameter can be set in the model’s Properties dialog. See “Non-Conservative Nodes Tab” on page 153 for more information. To add a probe Display Element to a sheet, right-click a model on the sheet and select «Probe».

Outputs in Properties Dialog

A simulation quantity of a model can be made available for access online or in a file from the Properties dialog of a model on the sheet. On the «Output/Display» tab, check the Online box for a quantity to make it available for viewing on the sheet. Check the File box to store the quantity in the database.

Outputs in Display Elements

Display Elements allow the viewing of all simulation quantities of a sheet as online output. Place a Display Element on the sheet. Right-click and open the Properties dialog. On the «Channels» tab, open the simulation quantity tree and select the simulation quantities to be displayed. Only a model’s defined simulation quantities can be selected in Display Elements for display on the sheet. Each selected quantity can have color, displayed number component (real, imaginary, magnitude, or phase), Y-axis, scale, and offset set for it. In the simulation quantity tree, model inputs are shown in red, model outputs are shown in blue, and model in/
VHDL-AMS Models in SIMPLORER

out quantities, as well as locally defined values, are shown in brown.

A quick explanation of how to set up a 2D View graph follows. To learn about setting up a more complex Display Element, read the next section, which contains a description of the 2D Digital Graph Display Element.

1. Select the Displays tab of the ModelAgent. Choose the SIMPLORER model library Displays. Click once in the ModelAgent window on the Displays tab. The window underneath shows the elements of the library in a tree structure.

2. Place the element on the sheet. Select the folder Displays from the model tree. Click on the “+” symbol to open the folder and display its contents. Select the element 2D View. Hold the mouse button down, drag the component to a free space on the sheet, and release the mouse button.

3. Define the model outputs. Double-click the 2D View symbol to open the Properties dialog. Select TR (Run data) from the drop-down menu on the «Channels» tab. In the Y-Axis section, create new outputs by clicking the Add button and checking the output boxes of the available quantities. Click <OK> to apply the changes.

4. Define the representation of the Display Elements and the output quantities. Change the size of a Display Element by selecting it and dragging the blue sizing handles with the cursor. To move a Display Element, select it, hold the mouse button down, and drag the Display Element to a new location. Change the color or other characteristics of the output quantities by selecting a quantity from the list on the «Channels» tab and changing the display presentation.

Objects can be edited within the Display Elements directly. Right click on the Display Element and select «Edit In Place». To move a component of the Display Element, CTRL-click the component and drag it to a new location. To edit a component of the Display Element, double-click it to open a Properties dialog where the font, color, and so on can be modified. Components include the legend, title, axis labels, and individual graph plots.
2D Digital Graph

The 2D Digital Graph and 2D Digital Graph (con) are especially designed to display digital signals of VHDL-AMS models. In contrast to the 2D View, each signal has its own coordinate system. The 2D Digital Graph can show vector quantities, special display formats, and data types. To show and hide members of a vector, open the «Extern Views» and click the + sign in the diagram legend. The 2D Digital Graph has a special cursor display in the Extern View.

Double-clicking the Display Element opens its Properties dialog and provides access to the tabs described in the following sections.

Channels Tab

The assignment of simulation quantities for the X and Y axes is defined on the «Channels» tab. All selected simulation quantities, the system time t, and the frequency f for both X and Y axes are available.

Depending on the data type and the display format, different data formats for representation can be specified. The decimal number 100 would be represented as 64 in hexadecimal format, as 144 in octal format, as 0110 0100 in binary format, and as d in ASCII format. Zoom in on the graphic when literals are not displayed in the view.

To define a marker, representation type, and line of the curve, click <Settings>. A scale and shift factor in Y-direction can also be entered for each quantity.
**X-Axis Tab**

**Sliding Window**
The Sliding Window defines the maximum display of the X-range. The window will be moved during the simulation between the simulation start and simulation end time.

**Presentation**
These settings define position, color, and the font of the inscription of the X-axis.

**Scale**
The adaptation of presentation areas for the X- and Y-axes can be performed manually or automatically. If automatic scaling is selected, the simulation quantities are displayed at maximum size. The dimension defines the maximum extended channel. If automatic values are not selected, the values entered for the axes scales are valid.

**Grid Lines**
Grid lines can be displayed or hidden in the diagram, and the minimum number of grid lines can be set.

**Data Format**
Three different formats (12.3k, 12.3E+3, or 1.23E+4) are available in the list, and the number of decimal places to be displayed with the fixed mantissa value can be specified.

**Y-Axis Tab**

The Presentation, Grid Lines, and Data Format tab functions are the same as for the X-axis. See “X-Axis Tab” on page 24.

**Presentation Tab**
The Presentation tab defines all properties of the diagram’s appearance. If a check box is selected, the corresponding option is active. The name, defined in the text box at the top of the tab, is used as the title of the Display Element on the sheet.

**Graph Border**
The graph border settings define the space between outer and inner frames of the display, as well as the diagrams, in percent. The value in the «Member» box sets the distance between members of a vector.

**Diagram Height**
The values define the height of the corresponding diagram type (analog, digital, event, literal) in percent.

**Color**
The colors in the dialog define the presentation of those items on the model sheet. To change a color, click the corresponding color box to open the color menu.

**Color Scheme**
Schematic also offers predefined color schemes or the option to save user-defined color schemes. To save a color scheme, set the colors in the dialog, type a new name for the scheme in the input field, and click <Save>. Load an existing scheme by selecting a name from the list and clicking <Load>. 
3 Automotive Powernet System Example

This basic automotive powernet example describes the development of a simulation model for an automotive powernet in six incremental steps. In the first step, a current source controlled by speed is used to model an alternator and introduces multidomain modeling concepts. The second step of the example introduces a simple behavioral 14V battery model with state of charge calculations that can be used in parallel with the alternator model in the powernet. The third step of the example adds a 42V subnet to the existing 14V powernet with a DC-DC converter model that supplies a 42V admittance load. The fourth step of the example introduces an ignition switch to supply different types of loads such as a switched load, a power load, and a lamp load. The fifth step illustrates the use of configuration files. The sixth step illustrates how different speed profile characteristics from text files can be used as different architectures for the powernet model.

The examples used in this chapter were derived from a powernet design developed by the MSR Consortium [1]. The models used in the examples are highly simplified to be better suited for the introduction of VHDL-AMS concepts.

This chapter contains information on:

- Model overview about automotive powernet system example
- Using example sheets on CD
- VHDL-AMS modeling features described in these steps
  - Step 1: Alternator that transforms speed input to electrical output
  - Step 2: Battery with state of charge calculation
  - Step 3: DC-DC Converter that supplies a 42V admittance load
  - Step 4: Ignition switch to supply different types of loads
  - Step 5: Configuration files
  - Step 6: Powertrain with different speed profiles from text files
Automotive Powernet System Example

Model Overview

The two main sources of electrical power in a vehicle are the alternator and the battery. The alternator transforms the mechanical power of the engine into electrical power, which is then distributed to the powernet. When a car is started, the engine takes some time to power up with the initial power being provided by the battery. Once the engine is up and running, the alternator supplies all the power required by the powernet. When the electric power supplied by the alternator is less than that required by the vehicle, the battery is used as a supplemental source. At other times when the power generated by the alternator is greater than the vehicle's power requirements, the extra power can be used to charge the battery. This simplified block diagram shows the main components of an automotive powernet system.

- The powertrain block is used to describe the speed profile of the vehicle. In this example, the powertrain describes the speed profile of the engine over a period of time.
- The alternator and the lead-acid battery supply the electrical power to loads that typically operate off a 14V bus. In this example, the loads operate from a 12V battery, and there is a dual bus system with both 14V and 42V.
- The DC-DC converter transforms 14V to 42V and supports the 42V bus.
- Three loads are present on the 14V system voltage bus and one load on the 42V system voltage bus.
- The ignition switch is present for the 14V system bus, while the 42V bus is always powered (keyless bus).
- The control panel is used to switch on/off or trigger the loads in the car. The control panel can be compared to a user switching on and off the AC/seat warmer/GPS module/Wiper system, etc. controls on the dashboard of the car.
- The non-ideal wires of the powernet are modeled as wires with given resistances. Typically, wire harnesses with specific characteristics are used for the powernet connections.
Using Example Sheets

The project files for the examples described in this chapter and the next, and the required library `vhdlams_tutorial.smd`, are provided on the student version CD. SIMPLORER 7.0 is needed to create and simulate the examples. The Student Version has some limitations in the number and types of simulation models available. If an example cannot be executed with the student version, a note is included on the sheet in the project file. In order to run these examples with the student version, some models need to be excluded from the simulation. To do this, select the models that need to be excluded and choose ELEMENT>DON'T ADD TO MODEL DESCRIPTION. The selected models will be hatched, indicating that they are not included in the simulation.

The project files for the examples can be opened from the SIMPLORER SCC or created by following the step-by-step instructions in this chapter. All sheets for the powernet example can be accessed from the `system_examples.ssc` project file, which can be found in the Tutorial Examples folder. The steps of the example can be done sequentially, extending the example sheets step-by-step, or starting with any step since each step explains how to create the complete example. If there are no instructions to change parameter values in the examples, the default values are used.

Creating Examples from Scratch

To create the examples in this chapter from scratch, start SIMPLORER and create a new project as explained in “Creating a New Project and Starting SIMPLORER Schematic” on page 6. Once the project is created and Schematic is open, click on the «Projects» tab in the ModelAgent (it should be empty). Right-click in the tab and choose «Insert Library». Navigate to the Tutorial Examples folder and select `vhdlams_tutorial.smd`. The models from the tutorial library should appear in the «Projects» tab.

Each of the examples in this chapter includes a schematic that shows how the models are connected, and step by step instructions explaining:

- Which models to place on the sheet
- Parameter values that need to be set for the models
- Architecture changes that need to be made
- Display Elements to be added to the sheet, with any presentation settings changes

Each example also includes an in-depth discussion of the entity and architecture descriptions for critical models in the system, with an explanation of the VHDL-AMS code contained in those models.

Each example is an extension of the previous example, adding models to the existing schematic.
## VHDL-AMS Modeling Features

The following table lists VHDL-AMS features and models used in the several steps of the Automotive Powernet System example and indicates if the examples can be run in the Student Version (SV).

<table>
<thead>
<tr>
<th>Step</th>
<th>Modeling Concepts</th>
<th>Model</th>
<th>SV</th>
</tr>
</thead>
</table>
| 1    | • Multidomain modeling  
      • *CONSTANT* and *FUNCTION* statements | Alternator           | yes |
| 2    | • *BREAK* and *IF-USE* statements  
      • ’*DOT*’ attribute | Battery              | yes |
| 3    | • Modeling with OmniCasters  
      • Solvability Criteria  
      • ’*INTEG*’ attribute | DC-DC Converter      | partly |
| 4    | • Mixed-signal modeling  
      • *SIGNAL* statement | Ignition switch      | partly |
| 5    | • Modeling with several configurations and architectures  
      • *PROCESS* statement | Control Panel        | no  |
| 6    | • *FILE* I/O statements  
      • Multidomain modeling using Domain-to-Domain model | Powertrain           | yes |
3.1 Step 1: Alternator

The first step in the powernet example, found in the file system_alternator.ssh, introduces concepts in multidomain modeling with a simple alternator model. The alternator is modeled as a current source that transforms speed input to electrical output.

The powertrain block provides the speed profile of the engine through a mechanical pin output. This mechanical pin is connected as an input to the alternator model (all mechanical connections are purple in the figure). The alternator model provides the electrical power to the entire system.

The following figure shows the `system_alternator.ssh` sheet. The `cyc_60` architecture is used for the powertrain model to provide a drive cycle for a period of 60 sec.

This example does not provide comprehensive results for all possible speed profiles. It is used to introduce the general modeling design that will be completed in the six modeling steps of the Powernet System Example.

Creating the Simulation Model

1. **Place and arrange all components shown in the figure above.** Alternator, Powertrain, and Resistor are VHDL-AMS models from the library on the «Projects» tab. Choose CONNECT>GND or type CTRL+G to place the ground node.

2. **Connect the models.** Place the cursor on one of the model pins to get the wire cursor and click to enter wire mode. Connect the components as indicated in the figure, setting the beginning, the corners, and the end of a wire with the mouse. Press ESC to end wire mode.

3. **Define the resistor parameters.** Double-click the resistor symbol to open the Properties dialog. Change the name from `res1` to `r_load` (The component name can be changed by clicking on it, entering the new name, then pressing ENTER.) Click in the «Value» field, and enter `1.0m` for the resistance value. Click <OK> to apply the changes.

4. **Place and arrange a 2D Display Element to display simulation results.** The ModelAgent «Displays» tab provides Display Elements to show simulation outputs on the sheet. Select the 2D View element, and drag it on the sheet.

5. **Open the simulation quantities list.** Double-click the Display Element to open the Properties dialog. Click the Add symbol to open the list of simulation quantities. This process can take some seconds because all quantities of the model sheet must be scanned.

6. **Select the simulation quantities.** Check the output boxes of `alternator1.current_out`, `alternator1.speed_out`, and `r_load.v`. Click <OK> to add these quantities to the list of displayed quantities.
7 Specify the display presentation settings. Select one of the output quantities in the list and change the display presentation settings as necessary. Note the scaling used for the graphs shown in “Results” on page 34. Click <OK> to apply the changes.

8 Define the simulation parameters. Choose SIMULATION>PARAMETERS, and change the default value for Simulation End Time to 60 sec, Minimum Time Step to 1 m sec, and Maximum time step to 10 m sec. Set Integration Formula to «Trapezoid». Click <OK> to apply the changes.

9 Save the sheet. Choose FILE>SAVE AS, enter a file name and directory, and click <OK>.
Overview of Parameter Values

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Parameter Values</th>
<th>Tab/Library</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>powertrain1</td>
<td>no parameters</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>cyc_60</td>
</tr>
<tr>
<td>alternator1</td>
<td>no parameters</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>r_load</td>
<td>r_nom [Ω]=1m</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
</tbody>
</table>

Alternator Model

The alternator model is a multidomain model that uses the electrical (ELECTRICAL) and mechanical (ROTATIONAL_V) domains; consequently, the electrical_systems and mechanical_systems packages that are defined in the IEEE library must be used. (To view the entity and architecture descriptions for a model, double-click on the component name in the ModelAgent.) The following statements in the model definition include the two packages:

```
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MECHANICAL_SYSTEMS.ALL;
```

Entity Description

The model accepts the speed input from a speed source through a ROTATIONAL_V terminal and outputs the corresponding electrical current and voltage through two ELECTRICAL terminals. It also provides the speed, torque, and current values as outputs from the model. The following table summarizes the port definitions in the model interface:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Name</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERMINAL</td>
<td>rot_in</td>
<td>ROTATIONAL_V</td>
</tr>
<tr>
<td></td>
<td>p_out</td>
<td>ELECTRICAL</td>
</tr>
<tr>
<td></td>
<td>m_out</td>
<td>ELECTRICAL</td>
</tr>
<tr>
<td>QUANTITY</td>
<td>speed_out</td>
<td>OUT VELOCITY</td>
</tr>
<tr>
<td></td>
<td>current_out</td>
<td>OUT CURRENT</td>
</tr>
<tr>
<td></td>
<td>torque_out</td>
<td>OUT TORQUE</td>
</tr>
</tbody>
</table>

The equivalent VHDL-AMS description for defining the model interface is as follows:

```
ENTITY Alternator IS
  PORT (
    rot_in : ROTATIONAL_V;
    p_out, m_out : ELECTRICAL;
    speed_out : OUT VELOCITY;
    current_out : OUT CURRENT;
    torque_out : OUT TORQUE);
END ENTITY Alternator;
```

The following figure shows the equivalent model interface of the alternator in the Schematic.
Architecture Description

The model’s architecture describes the method in which the alternator transforms the mechanical energy to electrical energy. This alternator is a simple model assumed to have no inertia; therefore, the torque depends on the input speed and the electrical power output. The alternator uses a characteristic function to obtain an output current value based on the input speed as shown in the following table:

<table>
<thead>
<tr>
<th>Speed (rpm)</th>
<th>0</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>6000</th>
<th>10000</th>
<th>14000</th>
<th>\infty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current (A)</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>60</td>
<td>70</td>
<td>85</td>
<td>90</td>
<td>100</td>
<td>105</td>
<td>105</td>
</tr>
</tbody>
</table>

The model defines a constant value of $\Omega/2N$ that is equal to $60/2\pi$ for transforming the value of rotational speed from rad/s to rpm. The VHDL-AMS statement for defining the constant value is as follows:

```vhdl-ams
CONSTANT $\Omega/2N$ : REAL := 60.0/MATH_2_PI;
```

The model calculates a torque output based on the following equations:

- Torque $\times$ Omega $=$ Voltage $\times$ Current
- Torque $=$ Voltage $\times$ Current/Omega

The equivalent VHDL-AMS description for defining the model architecture is as follows:

```vhdl-ams
ARCHITECTURE behav OF Alternator IS
  CONSTANT $\Omega/2N$ : REAL := 60.0/MATH_2_PI;  -- This is equal to 60/2*pi
  QUANTITY v ACROSS i THROUGH p_out TO m_out;
  QUANTITY omega ACROSS torque THROUGH rot_in TO ROTATIONAL_V_REF;
  FUNCTION CH (x: REAL) RETURN real IS
    CONSTANT X1 : REAL := 1000.0 ;
    CONSTANT Y1 : REAL := 0.0  ;
    CONSTANT X2 : REAL := 1500.0 ;
    CONSTANT Y2 : REAL := 30.0  ;
    CONSTANT X3 : REAL := 2000.0 ;
    CONSTANT Y3 : REAL := 60.0  ;
    CONSTANT X4 : REAL := 3000.0 ;
    CONSTANT Y4 : REAL := 70.0  ;
    CONSTANT X5 : REAL := 4000.0 ;
    CONSTANT Y5 : REAL := 85.0  ;
    CONSTANT X6 : REAL := 6000.0 ;
    CONSTANT Y6 : REAL := 90.0  ;
    CONSTANT X7 : REAL := 10000.0 ;
    CONSTANT Y7 : REAL := 100.0 ;
    CONSTANT X8 : REAL := 14000.0 ;
    CONSTANT Y8 : REAL := 105.0 ;
    VARIABLE result: REAL := 0.0 ;
```

```vhdl-ams
```
BEGIN
IF X < X1 THEN result:= Y1;
ELSE IF X < X2 THEN result:= Y1+(x - X1) / (X2-X1) * (Y2-Y1);
ELSE IF X < X3 THEN result:= Y2+(x - X2) / (X3-X2) * (Y3-Y2);
ELSE IF X < X4 THEN result:= Y3+(x - X3) / (X4-X3) * (Y4-Y3);
ELSE IF X < X5 THEN result:= Y4+(x - X4) / (X5-X4) * (Y5-Y4);
ELSE IF X < X6 THEN result:= Y5+(x - X5) / (X6-X5) * (Y6-Y5);
ELSE IF X < X7 THEN result:= Y6+(x - X6) / (X7-X6) * (Y7-Y6);
ELSE IF X < X8 THEN result:= Y7+(x - X7) / (X8-X7) * (Y8-Y8);
ELSE result:=Y8;
END IF;
END IF;
END IF;
END IF;
END IF;
RETURN result;
END FUNCTION CH;
BEGIN
i == -CH(omega * OM2N);
torque == -i*v / (omega + 1.0E-12);
speed_out == omega*OM2N;
torque_out == torque;
current_out == i;
END ARCHITECTURE behav;

Usually, subprogram modules like FUNCTION declarations and reusable declarations such as TYPE/CONSTANT declarations are defined in a package. In this model, the function and constant definitions are within the model architecture since they need not have a scope outside of the model.
Start the simulation using the SIMULATION>START menu command or the F12 key. The simulation model is compiled and evaluated. The Display Element shows the simulation results on the sheet. The alternator outputs an electrical current based on the input speed from the powertrain block.
3.2 Step 2: Battery

The second step of the example, found in the file system_battery.ssh, adds a simple behavioral 14V battery model that can be used in parallel with the alternator model in the powernet. The alternator and battery models provide the electrical power to the entire system. The battery characteristics of interest are the current, voltage, and the State of Charge (SoC). The SoC of the battery is expressed as a value between '0' and '1' and suggests how much charge from the battery is used up during a drive cycle. When the alternator is able to support the loads and drive current into the battery, the battery SoC increases. On the other hand, if the battery is powering the electrical system of the vehicle, its SoC decreases. The graph of the SoC indicates how the SoC varies continuously as different loads are switched on and off.

The following figure shows the extended system_alternator.ssh sheet including the battery model and the wire component.

Creating the Simulation Model

1. **Place and arrange all components shown in the figure above.** (The schematic from the previous step can be extended.) Alternator, Powertrain, Battery, and Wire are VHDL-AMS models from the library on the «Projects» tab. The ammeter is a model from the Measurement folder of the Basic vhdlams.smd library on the «AMS» tab. Type CTRL+G to place the ground node.

2. **Connect the models.** Place the cursor on a model pin to get the wire cursor and click to enter wire mode. Connect the components as indicated in the figure, setting the beginning, the corners, and the end of a wire with the mouse. Press ESC to end wire mode.

3. **Define the wire component parameters.** Double-click the wire component to open the Properties dialog. Click in the «Value» field, and enter 3m for the resistance value. Click <OK> to apply the changes.

4. **Place and arrange two 2D Display Elements to display simulation results.** The ModelAgent «Displays» tab provides Display Elements to show simulation outputs on the sheet. Drag two 2D View elements onto the sheet.

5. **Open the simulation quantities list on one of the 2-D View elements.** Double-click the Display Element to open the Properties dialog. Click the Add symbol to open the list of simulation quantities. This process can take some seconds, because all quantities of the model sheet must be scanned.

6. **Select the simulation quantities.** Check the output boxes of alternator1.speed_out, bat_soc.v_out, and am1.i. Click <OK> to add these quantities to the list of displayed quantities.
7 Change the display presentation settings. Select each output quantity in the list and change the display presentation settings as necessary. Note the scaling used in “Results” on page 39. Click <OK> to apply the changes.
Repeat the steps 5, 6, and 7 for the second Display Element, but check the box of bat_soc.soc_out as output.

8 Define the simulation parameters. Choose SIMULATION>PARAMETERS and change the default value for Simulation End Time to 60 sec, Minimum Time Step to 1m sec, and Maximum Time Step to 10m sec. Set Integration Formula to “Trapezoid”. Click <OK> to apply the changes.

9 Save the sheet. Choose FILE>SAVE AS, enter a file name and directory, and click <OK>.

Overview of Parameter Values

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Parameter Values</th>
<th>Tab/Library</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>powertrain1</td>
<td>no parameters</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>cyc_60</td>
</tr>
<tr>
<td>alternator1</td>
<td>no parameters</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>bat_soc1</td>
<td>factor=1; v_init[V]=12; v_max[V]=14</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>wire1</td>
<td>r_nom [Ω]=3m</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>sm1</td>
<td>no parameters</td>
<td>AMS/basic_vhdlams.smd</td>
<td>behav</td>
</tr>
</tbody>
</table>

Battery Model

The battery model uses the electrical (ELECTRICAL) domain; consequently, the electrical_systems package that is defined in the IEEE library must be included. The following statements in the model definition include the package:

```
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
```

Entity Description

The entity description of the battery model uses static GENERIC parameters (constant value inputs evaluated only at the beginning of the simulation) and terminals in the interface. The battery model has three parameters: factor, initial voltage, and maximum voltage, as defined in the GENERIC statement. The parameter factor refers to a scaling factor that can scale the capacitance values. The battery provides its electrical output through a pair of electrical terminals, the voltage across the battery as an output through v_out, and the state of charge of the battery through soc_out as defined in the QUANTITY statement.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Name</th>
<th>Property</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC</td>
<td>factor</td>
<td>REAL</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>v_init</td>
<td>VOLTAGE</td>
<td>12.0</td>
</tr>
<tr>
<td></td>
<td>v_max</td>
<td>VOLTAGE</td>
<td>14.0</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>p</td>
<td>ELECTRICAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m</td>
<td>ELECTRICAL</td>
<td></td>
</tr>
<tr>
<td>QUANTITY</td>
<td>v_out</td>
<td>VOLTAGE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>soc_out</td>
<td>REAL</td>
<td></td>
</tr>
</tbody>
</table>
The equivalent VHDL-AMS description for defining the model interface is as follows:

```vhdl-ams
ENTITY bat_soc IS
  GENERIC(
    factor: REAL := 1.0;
    v_init : VOLTAGE := 12.0;
    v_max : VOLTAGE := 14.0);
  PORT(
    TERMINAL p,m : ELECTRICAL;
    QUANTITY soc_out : OUT REAL := 0.0;
    QUANTITY v_out : OUT VOLTAGE := 0.0);
END ENTITY bat_soc;
```

The following figure shows the equivalent model interface of the Battery model in the Schematic:

**Architecture Description**

In a lead-acid battery, chemical energy is transformed to electrical energy at the battery electrodes through chemical reaction of the acid molecules with the electrode materials. The chemical reaction at the battery electrodes results in a lower concentration of the lead acid near the electrodes during discharge.

The diffusion of the acid molecules between regions of high and low concentration is modeled with three components: the diffusion resistor, a 'slow' capacitor, and a 'fast' capacitor. Since the acid near the battery electrodes is consumed faster, its concentration is lower. This is modeled as a fast capacitor that discharges quickly. The acid concentration farther from the electrodes is not consumed as fast, so it is modeled as a 'slow' capacitor that discharges slowly. The actual diffusion of the acid molecules between the regions of low and high acid concentrations is modeled with a diffusion resistor. The electrical resistance of the electrodes is modeled as the internal resistance of the battery.

The `behav` architecture of the battery model describes the behavioral implementation of the following circuit:
The internal resistance \( r_i = 10 \text{m} \), diffusion resistance \( r_d = 40 \text{m} \), fast capacitor \( c_f = 60 \times \text{factor} \), and slow capacitor \( c_s = 20000 \times \text{factor} \) are constant values. The model evaluates the circuit based on the following four equations:

\[
\begin{align*}
\text{v}_i &= i_r \times r_i, \quad \text{d}(\text{v}_f) / \text{dt} = 1 / (fc \times \text{factor}) \times i_f \\
\text{v}_d &= i_r \times r_d, \quad \text{d}(\text{v}_s) / \text{dt} = 1 / (sc \times \text{factor}) \times i_s
\end{align*}
\]

The equivalent VHDL-AMS description for defining the model architecture is as follows:

```vhdl-ams
ARCHITECTURE behav OF bat_soc IS
TERMINAL t1, t2: ELECTRICAL;
CONSTANT ri: RESISTANCE := 1.0e-2;
CONSTANT fc: CAPACITANCE := 60.0;
CONSTANT rd: RESISTANCE := 4.0e-2;
CONSTANT sc: CAPACITANCE := 2.0e4;
CONSTANT init_charge : CHARGE := ((fc + sc) \times \text{factor}) \times \text{v_init};
CONSTANT max_charge : CHARGE := ((fc + sc) \times \text{factor}) \times \text{v_max};
QUANTITY v_ri ACROSS i_ri THROUGH p TO t1;
QUANTITY v_fc ACROSS i_fc THROUGH t1 TO t2;
QUANTITY v_rd ACROSS i_rd THROUGH t1 TO t2;
QUANTITY v_sc ACROSS i_sc THROUGH t2 TO t1;
QUANTITY \text{v} ACROSS p TO m;
QUANTITY \text{ri_val} : \text{RESISTANCE} := \text{ri};
QUANTITY total_charge : \text{CHARGE} := \text{init_charge};
BEGIN
BREAK v_fc => \text{v_init}, v_sc => \text{v_init};
BREAK i_ri => \text{v_init}/ri;
BREAK soc_out => init_charge/max_charge;
IF (i_ri < 0.0) OR (soc_out < 0.1) USE
  ri_val = (2.0*ri)**2/(1.0-soc_out);
ELSE
  ri_val = (2.0*ri)**2/soc_out;
END USE;
v_ri == i_ri \times ri_val;
total_charge == ((fc*factor) \times v_fc) + ((sc*factor) \times v_sc);
soc_out == total_charge/max_charge;
v_fC' == 1.0/(fc*factor) \times i_fC;
v_rd == i_rD \times rd;
v_sc'dot == 1.0/(sc*factor) \times i_sc;
v_out == v;
END ARCHITECTURE behav;
```

The state of charge of the battery is based on the initial and maximum voltage values of the battery.

The \textbf{IF-USE} statement is used to limit the charging of the battery when its SoC value approaches '1', and the internal resistor is used to limit the discharging of the battery when its SoC value approaches '0'. Unrealistic values of SoC can occur by forcing/drawing large currents to/from the battery.

The \textit{'DOT'} attribute is used to evaluate the derivative of an analog quantity in a simultaneous differential algebraic equation.

To model the discontinuous behavior of the battery model, \textbf{BREAK} statements are used to indicate the occurrence of a discontinuity to the analog solver. Here, since the first derivative of the \textit{v}_fc and \textit{v}_sc quantities may be discontinuous the \textbf{BREAK} statement uses initial voltage values that are specified by \textit{v}_init.
Results

Start the simulation using the SIMULATION->START menu command or the F12 key. The simulation model is compiled and evaluated. The Display Elements show the simulation results on the sheet.

The alternator outputs an electrical current based on the input speed from the Powertrain model. The State of Charge (SoC) of the battery is expressed as a value between ‘0’ and ‘1’. When the alternator drives current into the battery, the SoC of the battery increases. If the battery is powering the electrical system of the vehicle, its SoC decreases.
3.3 Step 3: DC-DC Converter

The third step of the example, found in the file system_dcdc.ssh, adds a 42V subnet to the existing 14V powernet with a DC-DC converter model that supplies a 42V admittance load. The system voltage of 14V is stepped-up to 42V by a DC-DC converter to provide the dual bus voltage. The 42V bus supplies an admittance load that is switched on and off by a control signal with a triangular waveform characteristic. The use of OmniCaster models is illustrated by connecting a REAL QUANTITY output of a triangular time function model with the REAL SIGNAL input required by the admittance load. If the pins of these models are connected directly, a flexible OmniCaster will automatically be inserted between them. The flexible OmniCaster model will choose the correct transformation based on the data types of the pins connected to it. Or, the flexible OmniCaster model (found on the «Tools» tab of the ModelAgent) or the Real-Real OmniCaster model (from the Quantity-Signal folder under the OmniCaster folder on the «Tools» tab) can be placed on the sheet and connected as shown in the figure below.

The following figure shows the system_battery.ssh sheet extended with the DC-DC model and the 42V subnet:

Creating the Simulation Model

1. Place and arrange all components shown in the figure above. (The schematic from the previous step can be extended.) Alternator, Powertrain, Battery, Capacitor, DC-DC Converter, Load, and Wire are VHDL-AMS models from the library on the «Projects» tab. To insert the OmniCaster model manually, place it on the sheet. The Triangular Wave is a model from the Time Functions folder under the Tools folder of the basic_vhdams library on the «AMS» tab. Type CTRL+G to place a ground node.

2. Connect the models. Place the cursor on a model pin to get the wire cursor and click to enter wire mode. Connect the components as indicated in the figure, setting the beginning, the corners, and the end of a wire with the mouse. Press ESC to end wire mode.

3. Define wire parameters. For each wire model, double-click the wire symbol to open the Properties dialog. Click in the «Value» field, and enter the corresponding resistance value. Click <OK> to apply the changes.

4. Define parameters of the capacitor. Double-click the capacitor symbol to open the Properties dialog. Click in the «Value» field, and enter 10m as capacitance. Click <OK> to apply the changes.
5 Define parameters of the load. Double-click the load symbol to open the Properties dialog and change the model name from load1 to load_42. The model uses the admittance architecture (default setting). Click <OK> to apply the changes.

6 Define parameters of the triangular wave. Double-click the triangular wave model to open its Properties dialog. Click in the «Value» fields and enter 0.05 as offset, 0.05 as amplitude, and 0.2 as frequency value. Click <OK> to apply the changes.

7 Place and arrange two 2D Display Elements to display simulation results. The ModelAgent «Displays» tab provides Display Elements to show simulation outputs on the sheet. Drag two 2D View elements onto the sheet.

8 Open the simulation quantities list on one of the 2-D View elements. Double-click the Display Element to open the Properties dialog. Click the Add symbol to open the list of simulation quantities. This process can take some seconds, because all quantities of the model sheet must be scanned.

9 Select the simulation quantities. Check the output boxes of load_42.i, load_42.v, and load_42.ctrl_ramp. Click <OK> to add these quantities to the list of displayed quantities.

10 Change the display presentation settings. Select each output quantity in the list and change the display presentation settings as necessary. Note the scaling used in the Display Elements shown in "Results" on page 44. Click <OK> to apply the changes. Repeat the steps 8, 9, and 10 for the second Display Element but check the box of bat_soc1.soc_out as output.

11 Define the simulation parameters. Choose SIMULATION>PARAMETERS and change the values for Simulation End Time to 5 sec, Minimum Time Step to 1m sec, Maximum Time Step to 1 sec, and Maximum Number of Iterations to 45. Set Integration Formula to «Trapezoid». Click <OK> to apply the changes.

12 Save the sheet. Choose FILE>SAVE AS, enter a file name and directory, and click <OK>.

Overview of Parameter Values

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Parameter Values</th>
<th>Tab/Library</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>powertrain1</td>
<td>no parameters</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>cyc_60</td>
</tr>
<tr>
<td>alternator1</td>
<td>no parameters</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>bat_soc1</td>
<td>factor=1; v_init[V]=12;</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td></td>
<td>v_max[V]=14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wire1</td>
<td>r_nom [Ω]=3m</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>wire2</td>
<td>r_nom [Ω]=1m</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>wire3</td>
<td>r_nom [Ω]=10m</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>wire4</td>
<td>r_nom [Ω]=10m</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>cap1</td>
<td>c_nom[F]=10m; v_init[V]=0</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>dcdc1</td>
<td>v_set[V]=42; i_in_max[A]=30</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>triang1</td>
<td>ampl=0.05; freq[Hz]=0.2;</td>
<td>AMS/basic_vhdlams.smd</td>
<td>behav</td>
</tr>
<tr>
<td></td>
<td>off=0.05; tdelay[s]=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OmniCaster1</td>
<td>ts[s]=1m, inp=trian1.val</td>
<td>Tools/transformations.smd</td>
<td>behav</td>
</tr>
<tr>
<td>load_42</td>
<td>p_nom[W]=1; v_nom[V]=14;</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>admittance</td>
</tr>
<tr>
<td></td>
<td>i_ramp[A]=10; on_ctrl=OmniCaster1.val</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
42 Automotive Powernet System Example

DC-DC Converter

The DC-DC converter uses the electrical (ELECTRICAL) domain; consequently, the electrical_systems package that is defined in the IEEE library must be included. The following statements in the model definition include the package:

```
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
```

Entity Description

The entity description of the DC-DC converter uses static parameters (constant value inputs evaluated only at the beginning of the simulation) and terminals in the interface. The DC-DC converter has two parameters for the set voltage and maximum input current as indicated in the GENERIC statement. These parameters are of type REAL with specific default values.

The model also has four terminals, a pair of plus (+) and minus (–) terminals at the input \((p_{in}, m_{in})\) and at the output \((p_{out}, m_{out})\). The following table summarizes the port definitions in the model interface:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Name</th>
<th>Property</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC</td>
<td>v_set</td>
<td>VOLTAGE</td>
<td>42.0</td>
</tr>
<tr>
<td></td>
<td>i_in_max</td>
<td>CURRENT</td>
<td>30.0</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>p_in</td>
<td>ELECTRICAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m_in</td>
<td>ELECTRICAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_out</td>
<td>ELECTRICAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m_out</td>
<td>ELECTRICAL</td>
<td></td>
</tr>
</tbody>
</table>

The equivalent VHDL-AMS description for defining the model interface is as follows:

```
ENTITY dcdc IS
  GENERIC:
    v_set: VOLTAGE := 42.0;
    i_in_max: CURRENT := 30.0;
  PORT:
    TERMINAL p_in, m_in, p_out, m_out : ELECTRICAL;
END ENTITY dcdc;
```

The following figure shows the equivalent model interface of the DC-DC converter in the Schematic:
Architecture Description

The *behav* architecture of the DC-DC converter uses a function to limit the current and voltage based on specified maximum and minimum values.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmin + ε ≤ x ≤ xmax - ε</td>
<td>x</td>
</tr>
<tr>
<td>xmin + ε ≥ x</td>
<td>xmin + ε²/(xmin-x+2*ε)</td>
</tr>
<tr>
<td>else</td>
<td>xmax - ε²/(x - xmax + 2*ε)</td>
</tr>
</tbody>
</table>

The model limits the voltage and current based on the following equations:

\[
\text{IF } x_{\text{min}} + \varepsilon \leq x \text{ AND } x \leq x_{\text{max}} - \varepsilon \text{ THEN RETURN } x
\]

\[
\text{ELSEIF } x_{\text{min}} + \varepsilon > x \text{ THEN RETURN } x_{\text{min}} + \varepsilon / (-x + x_{\text{min}} + 2.0 * \varepsilon)
\]

\[
\text{ELSE } x_{\text{max}} - \varepsilon / (x - x_{\text{max}} + 2.0 * \varepsilon)
\]

The equivalent VHDL-AMS description for defining the model architecture is as follows:

```vhdl-ams
ARCHITECTURE behav OF dcdc IS
  CONSTANT load_resistor: RESISTANCE := 0.1;
  CONSTANT tank_latency: REAL := 1.0;
  CONSTANT efficiency: REAL := 0.9;
  CONSTANT i_in_min: CURRENT := 0.0;
  CONSTANT v_tank_min : VOLTAGE := 10.0;
  CONSTANT p_ctrl : REAL := 1.0;
  CONSTANT i_ctrl : REAL := 1.0;
  CONSTANT v_tank_init: VOLTAGE := 12.0;
  QUANTITY v_in ACROSS i_in THROUGH p_in TO m_in;
  QUANTITY v_out ACROSS i_out THROUGH p_out TO m_out;
  QUANTITY v_tank, v_act_set, v_diff: VOLTAGE := 0.0;
  FUNCTION saturation( x, x_min, x_max: real) RETURN real IS
    CONSTANT eps: REAL := 1.0e-3;
    BEGIN
      IF x_min + eps <= x AND x <= x_max - eps THEN RETURN x;
      ELSIF x_min + eps >= x THEN RETURN x_min + eps / (-x + x_min + 2.0 * eps);
      ELSE RETURN x_max - eps / (x - x_max + 2.0 * eps);
      END IF;
    END FUNCTION;
    BEGIN
      BREAK v_tank => v_tank_init;
      i_in <= saturation(v_in - v_tank)/load_resistor, i_in_min, i_in_max);
      v_tank'dot == (i_in + i_out * v_out/v_tank)/efficiency/tank_latency;
      v_act_set == saturation(v_set, 0.0, v_tank * (v_set/v_tank_min));
      v_diff == v_act_set - v_out;
      i_out == -(p_ctrl * v_diff + i_ctrl * v_diff'integ);
    END ARCHITECTURE behav;
END ARCHITECTURE behav;
```

This model architecture illustrates the use of branch and free quantities in a model description. The statement

```vhdl-ams
QUANTITY v_in ACROSS i_in THROUGH p_in TO m_in;
QUANTITY v_tank, v_act_set, v_diff: VOLTAGE := 0.0;
```

associates two branch quantities, an across quantity *v*_in and a through quantity *i*_in, between the *p*_in and *m*_in terminals of the model. These branch quantities follow the energy conservation laws defined by nature of the terminals they are associated with. The statement

```vhdl-ams
QUANTITY v_tank, v_act_set, v_diff: VOLTAGE := 0.0;
```

defines three free quantities not associated with any laws of conservation, of type *VOLTAGE*. 
The solvability of a set of equations in a mixed signal model description should satisfy the following two rules:

- The number of equations specified in the model should be equal to the number of free quantities, through quantities, and port quantities of mode `OUT`.
  In this model description, the five equations specified account for three free quantities (`v_tank, v_act_set, v_diff`) and two branch quantities (`i_in, i_out`). There are no port quantities specified in the entity description.

- Any free quantities and any port quantities of mode `OUT` must appear in a simultaneous equation within the architecture of the model.
  In this model, `v_tank, v_act_set, and v_diff` all appear in simultaneous statements.

The equation for the output current models a PI controller and uses the `\text{INTEG}` attribute to obtain the implicit quantity value of $\int v\_diff \, dt$.

The Admittance load model is discussed with other load models in Chapter 4.4 “Multilevel Modeling Techniques: Loads” on page 90.

**Results**

Start the simulation using the SIMULATION>START menu command or the F12 key. The simulation model is compiled and evaluated. The Display Elements show the simulation results on the sheet.

Current and voltage of the Load model varies depending on the control signal and the resulting admittance (as the rate of change of the control signal).
3.4 Step 4: Ignition Switch and Loads

This step, found in the file system_ignition_loads.ssh, extends the powernet system example with three 14V load models and a digital control panel. The control panel provides the control inputs to each of the loads and controls the operation of the ignition switch that acts as a main control for the loads.

The following figure shows the extended system_dcdc.ssh sheet with the lamp and load models in the 14V subnet and the control panel.

The sheet can be simulated as shown with the full version of SIMPLORER. Since the student version limits the size of a design, some models need to be excluded from the simulation.

### Defining Simulation Model

1. **Place and arrange all components shown in the figure above.** (The schematic from the previous step can be extended.) Alternator, Powertrain, Battery, Capacitor, DC-DC Converter, Ignition Switch, Load, Lamp, Control Panel, and the Wire model are VHDL-AMS models from the library on the «Projects» tab. Type CTRL+G to place a ground node.

2. **Connect the models.** Place the cursor on a model pin to get the wire cursor and click to enter wire mode. Connect the components as indicated in the figure, setting the beginning, the corners, and the end of a wire with the mouse. Press ESC to end wire mode.

3. **Select the powertrain architecture.** Double-click the powertrain symbol to open the Properties dialog and click the «Library» tab. Choose the cyc_600 architecture from the list. This architecture provides a drive cycle waveform of a 600 second duration. Click <OK> to apply the changes.
4 Define wire parameters. Double-click each wire symbol to open its Properties dialog. Click in the \textit{Value} field, and enter the corresponding resistance value. Click <OK> to apply the changes.

5 Define parameters of the capacitor. Double-click the capacitor symbol to open the Properties dialog. Click in the \textit{Value} field, and enter $10\text{m}$ as capacitance. Click <OK> to apply the changes.

6 Define parameters of the loads. Double-click each load symbol to open its Properties dialog, and change the model name to the values shown in the diagram. Use the table in “Overview of Parameter Values” on page 47 to set the correct architecture for each load. Click <OK> to apply the changes.

7 Place and arrange two 2D Display Elements to display simulation results. The ModelAgent \textit{Displays} tab provides Display Elements to show simulation outputs on the sheet. Select one 2D View and one 2D Digital Graph, and drag them onto the sheet.

8 Open the simulation quantities list on the 2D Digital Graph. Double-click the Display Element to open the Properties dialog. Click the Add symbol to open the list of simulation quantities. This process can take some seconds, because all quantities of the model sheet must be scanned.

9 Select the simulation quantities. Check the output boxes of \texttt{control\_panel1.switch\_out}, \texttt{control\_panel1.lamp\_out}, \texttt{control\_panel1.power\_out}, \texttt{control\_panel1.switched\_out}, and \texttt{control\_panel1.admittance\_out}. Click <OK> to add these quantities to the list of displayed quantities.

10 Change the display presentation settings. Select each output quantity in the list and change the display presentation settings as necessary. Click <OK> to apply the changes.

Repeat the steps 8, 9, and 10 for the 2D View Display Element but check the box of \texttt{bat\_soc1.soc\_out} as the output. See also page 29.

11 Define the simulation parameters. Choose Simulation \textit{Parameters} and change the values for Simulation End Time to 400 sec, Minimum Time Step to $10\mu\text{sec}$, Maximum Time Step to 5 sec, Maximum Current Error to 0.01 and Maximum Voltage Error to 0.01. Set Integration Formula to \textit{Trapazoid}. Click <OK> to apply the changes.

12 Save the sheet. Choose File \textit{Save As}, enter file name and directory, and click <OK>.

The 2D Digital Graph is especially designed to display digital signals of VHDL-AMS models. In contrast to the 2D View, each signal has its own coordinate system (Y-axis).
### Overview of Parameter Values

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Parameter Values</th>
<th>Tab/Library</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>powertrain1</td>
<td>no parameters</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>cyc_600</td>
</tr>
<tr>
<td>alternator1</td>
<td>no parameters</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>bat_soc1</td>
<td>factor=1; v init[V]=12; v max[V]=14</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>wire1</td>
<td>r nom [Ω]=3m</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>wire2</td>
<td>r nom [Ω]=1m</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>wire3</td>
<td>r nom [Ω]=10m</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>wire4/5/6/7</td>
<td>r nom [Ω]=10m</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>cap1</td>
<td>c nom [F]=10m; v init[V]=0</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>dcdc1</td>
<td>v set[V]=42; i in max[A]=30</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>ignition1</td>
<td>on ctrl=control panel1.switch_out</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>load_42</td>
<td>p nom[W]=1; v nom[V]=14; t ramp[s]=10u; on ctrl=OmniCaster1.val</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>admittance</td>
</tr>
<tr>
<td>lamp</td>
<td>p nom[W]=20; v nom[V]=12; t ramp[s]=10u; on ctrl=control panel1.lamp_out</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>behav</td>
</tr>
<tr>
<td>sw_load</td>
<td>p nom[W]=1; v nom[V]=14; t ramp[s]=10u; on ctrl=control panel1.switched_out</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>switch</td>
</tr>
<tr>
<td>nom_load</td>
<td>p nom[W]=1; v nom[V]=14; t ramp[s]=10u; on ctrl=control panel1.power_out</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>nominal</td>
</tr>
<tr>
<td>control_panel1</td>
<td>no parameters</td>
<td>Projects/vhdlamsTutorial.smd</td>
<td>all_loads</td>
</tr>
</tbody>
</table>

### Ignition Switch Model

The ignition switch model is a simple non-ideal switch that accepts the on-resistance and off-conductance of the switch as parameters. A digital boolean signal with **TRUE/FALSE** values is used as a control signal to turn the switch on and off.

The Load model uses the electrical (ELECTRICAL) domain; consequently, the `electrical_systems` package that is defined in the IEEE library must be included. The following statements in the model definition include the package:

```
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
```
### Entity Description

This model defines two parameter values for `r_on` and `g_off`, terminals `p` and `m`, and a digital control signal `on_ctrl` of type `BOOLEAN`. The following table summarizes the port definitions in the model interface:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Name</th>
<th>Property</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC</td>
<td><code>r_on</code></td>
<td>RESISTANCE</td>
<td>1.0e-3</td>
</tr>
<tr>
<td></td>
<td><code>g_off</code></td>
<td>REAL</td>
<td>1.0e-9</td>
</tr>
<tr>
<td>TERMINAL</td>
<td><code>p</code></td>
<td>ELECTRICAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>m</code></td>
<td>ELECTRICAL</td>
<td></td>
</tr>
<tr>
<td>SIGNAL</td>
<td><code>on_ctrl</code></td>
<td>BOOLEAN</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

The equivalent VHDL-AMS description for defining the model interface is as follows:

```vhdl
ENTITY ignition IS
  GENERIC
    (r_on: RESISTANCE := 1.0e-3;
     g_off: REAL := 1.0e-9);
  PORT
    (TERMINAL p, m: ELECTRICAL;
     SIGNAL on_ctrl : IN BOOLEAN);
END ENTITY ignition;
```

The following figure shows the equivalent model interface of the Switch model in the Schematic:

---

### Architecture Description

The ignition switch model uses a simultaneous `IF` condition to turn the switch on/off based on the input boolean control signal, `on_ctrl`. The `BREAK` statement is used to ensure that the analog solver is able to resolve discontinuities that may arise because of the sudden transition of the digital control signal.

The equivalent VHDL-AMS description for defining the model architecture is as follows:

```vhdl
ARCHITECTURE behav OF ignition IS
  BEGIN
    IF on_ctrl USE
      v == i * r_on;
    ELSE
      i == v * g_off;
    END USE;
    BREAK ON on_ctrl;
  END ARCHITECTURE behav;
```
Results

Start the simulation using the SIMULATION>START menu command or the F12 key. The simulation model is compiled and evaluated. The Display Elements show the simulation results on the sheet.

Depending on the switching characteristics defined in the control panel the loads are switched on and off, causing the battery State of Charge to vary.
3.5 Step 5: Configuration Modes

This step in the powernet example, found in the file system_configurations.ssh, introduces the concept of configurations and illustrates how different configurations may be used to provide different environments or simulation scenarios for a system. Configurations include the binding information that specifies which architecture should be used for each component instantiation, and the parameter values for each instantiation.

Different simulation scenarios can be considered for different speed profile architectures and different control panel architectures. For example, one simulation scenario would be to simulate the system with a speed profile characteristic defined as a Manhattan drive cycle under conditions of low-current 12V loads and a 42V load. Another scenario would be to see how the system responds if it were provided with the UDDS (Urban Dynamometer Driving Schedule) drive cycle characteristics and used high current peak 12V loads only. These different simulation scenarios can be easily created, stored, and retrieved using configuration files.

The powertrain model has four architectures associated with it, which provide different speed profile characteristics: cyc_60, cyc_600, cyc_manhattan, cyc_udds (the VHDL-AMS descriptions for these architectures are discussed in detail in the sixth step of the automotive powernet system example). The Control Panel model has five architectures associated with it: all_loads, behav1 (low 12V loads, no 42V loads), behav2 (low 12V loads, high peaks, no 42V loads), behav3 (high 12V loads, no 42V loads), and behav4 (low 12V loads, with 42V load). The VHDL-AMS descriptions for these architectures are described in this step in more detail.

To create the example sheet see Chapter 3.4 Step 4: “Ignition Switch and Loads” on page 45. Change the model parameters as indicated in the next paragraph.

Change \( t_{\text{ramp}} \) to 1m for the lamp and the three loads. Change the values for Simulation End Time to 600 sec, Minimum Time Step to 1m sec, Maximum Time Step to 5 sec, Maximum Current Error to 0.01 and Maximum Voltage Error to 0.01. Set Integration Formula to «Trapezoid».

Add two new Display Elements from the ModelAgent «Displays» tab. Drag two 2D View elements onto the sheet. Set up one to display the Powertrain speed profile using parameter powertrain1.vel, and the other to display the Battery current using parameter wire2.i.
Control Panel

The control panel model provides the control signals for three load models, a lamp model, and the ignition switch.

Entity Description

The interface of this model provides control signals for the ignition switch, lamp model, switched load model, power load model, and admittance load model. The following table summarizes the port definitions in the model interface:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Name</th>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNAL</td>
<td>switch_out</td>
<td>BOOLEAN</td>
<td>Ignition Switch Control</td>
</tr>
<tr>
<td></td>
<td>lamp_out</td>
<td>REAL</td>
<td>Lamp Load Control</td>
</tr>
<tr>
<td></td>
<td>power_out</td>
<td>REAL</td>
<td>Power Load Control</td>
</tr>
<tr>
<td></td>
<td>switched_out</td>
<td>REAL</td>
<td>Switched Load Control</td>
</tr>
<tr>
<td></td>
<td>admittance_out</td>
<td>REAL</td>
<td>Admittance Load Control</td>
</tr>
</tbody>
</table>

The equivalent VHDL-AMS description for defining the model interface is as follows:

```vhdl
ENTITY control_panel IS
  PORT (  
    SIGNAL switch_out : OUT BOOLEAN := FALSE;  
    SIGNAL lamp_out  : OUT REAL := 0.0;  
    SIGNAL power_out : OUT REAL := 0.0;  
    SIGNAL switched_out : OUT REAL := 0.0;  
    SIGNAL admittance_out : OUT REAL := 0.0);  
END ENTITY control_panel;
```

Architecture Description

The architecture descriptions of the control panel model describe the control patterns for switching ON and OFF the loads and ignition switch of the powernet system. The five architectures associated with the control panel are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Lamp</th>
<th>Power</th>
<th>Switched</th>
<th>Admittance</th>
</tr>
</thead>
<tbody>
<tr>
<td>behav1</td>
<td>Low 12 V loads, No 42 V load</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>behav2</td>
<td>Low 12 V loads with high current peaks, No 42 V load</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>behav3</td>
<td>High 12 V loads with high current peaks, No 42 V load</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>behav4</td>
<td>Low 12 V loads, 42 V load</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>all_loads</td>
<td>Low 12 V loads with high current peaks, 42V load</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

The following table shows the VHDL-AMS description for all architectures:
ARCHITECTURE behav1 OF control_panel IS
BEGIN
PROCESS
BEGIN
switch_out <= TRUE;
WAIT;
END PROCESS;
PROCESS
BEGIN
lamp_out <= 0.6;
WAIT;
END PROCESS;
PROCESS
BEGIN
power_out <= 0.0;
WAIT;
END PROCESS;
END ARCHITECTURE behav1;

ARCHITECTURE behav2 OF control_panel IS
BEGIN
PROCESS
BEGIN
switch_out <= TRUE;
WAIT;
END PROCESS;
PROCESS
BEGIN
lamp_out <= 0.6;
WAIT;
END PROCESS;
PROCESS
BEGIN
power_out <= 0.0;
WAIT;
END PROCESS;
PROCESS
BEGIN
switched_out <= 0.0;
WAIT;
END PROCESS;
PROCESS
BEGIN
admittance_out <= 0.0;
WAIT;
END PROCESS;
END ARCHITECTURE behav2;

ARCHITECTURE behav3 OF control_panel IS
BEGIN
PROCESS
BEGIN
switch_out <= TRUE;
WAIT;
END PROCESS;
PROCESS
BEGIN
lamp_out <= 0.6;
WAIT;
END PROCESS;
PROCESS
BEGIN
power_out <= 0.0;
WAIT;
END PROCESS;
PROCESS
BEGIN
switched_out <= 0.0;
WAIT;
END PROCESS;
PROCESS
BEGIN
admittance_out <= 0.0;
WAIT;
END PROCESS;
END ARCHITECTURE behav3;

ARCHITECTURE behav4 OF control_panel IS
BEGIN
PROCESS
BEGIN
switch_out <= TRUE;
WAIT;
END PROCESS;
PROCESS
BEGIN
lamp_out <= 0.6;
WAIT;
END PROCESS;
PROCESS
BEGIN
power_out <= 0.0;
WAIT;
END PROCESS;
PROCESS
BEGIN
switched_out <= 0.0;
WAIT;
END PROCESS;
PROCESS
BEGIN
admittance_out <= 0.0;
WAIT;
END PROCESS;
END ARCHITECTURE behav4;
ARCHITECTURE all_loads OF control_panel IS
BEGIN
PROCESS
BEGIN
  switch_out <= FALSE;
  WAIT FOR 1 sec;
  switch_out <= TRUE;
  WAIT;
END PROCESS;
PROCESS
BEGIN
  WAIT FOR 200.0 sec;
  FOR i IN 1 TO 5 LOOP
    lamp_out <= 0.6;
    WAIT FOR 2.0 sec;
    lamp_out <= 0.0;
    WAIT FOR 2.0 sec;
  END LOOP;
  WAIT;
END PROCESS;
PROCESS
BEGIN
  power_out <= 0.0;
  WAIT FOR 100.0 sec;
  power_out <= 1.0;
  WAIT FOR 50.0 sec;
  power_out <= 0.0;
  WAIT;
END PROCESS;
PROCESS
BEGIN
  switched_out <= 0.0;
  WAIT FOR 300.0 sec;
  switched_out <= 1.0;
  WAIT FOR 50.0 sec;
  switched_out <= 0.0;
  WAIT;
END PROCESS;
PROCESS
BEGIN
  admittance_out <= 0.1;
  WAIT FOR 35000 ms;
  admittance_out <= 0.15;
  WAIT FOR 35000 ms;
  admittance_out <= 0.2;
  WAIT FOR 35000 ms;
END PROCESS;
END ARCHITECTURE all_loads;
Using Configuration Files

In this step of the system example, the following scenarios (configurations) for the powernet system are considered:

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Architectures</th>
<th>Control Panel</th>
<th>Configuration File</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cyc_manhattan</td>
<td>behav1</td>
<td>manhattan_config1.smc</td>
</tr>
<tr>
<td>2</td>
<td>cyc_udds</td>
<td>behav2</td>
<td>udds_config1.smc</td>
</tr>
<tr>
<td>3</td>
<td>cyc_udds</td>
<td>behav3</td>
<td>udds_config2.smc</td>
</tr>
<tr>
<td>4</td>
<td>cyc_manhattan</td>
<td>behav4</td>
<td>manhattan_config2.smc</td>
</tr>
</tbody>
</table>

Creating Configuration Files

To change the architecture of a model on the sheet, double-click the model to open the Properties dialog and click the «Library» tab. Choose the corresponding architecture from the list «Select Architecture/Modeling Level». Click <OK> to apply the changes. If a VHDL-AMS model from a library is placed on the sheet, the default architecture is always selected.

Set the architectures in the model Properties dialog for the first scenario. To save the configuration of the model architectures for the several scenarios, do the following:

1. Select the architectures for the Powertrain and the Control Panel for the first scenario.
2. Choose SIMULATION>START to run a simulation.
3. Choose FILE>SAVE CONFIGURATION, enter a file name, and click <OK>.
4. Repeat steps 1, 2, and 3 for the next three scenarios.

Loading Configurations

To load an existing configuration of model architectures, do the following:

1. Open example sheet system_configurations.ssh.
2. Choose FILE>LOAD CONFIGURATION.
3. Select an existing configuration file, for example manhattan_config1.smc.
4. Choose SIMULATION>START to run a simulation.

The architecture names of the models are automatically changed in the properties dialog of the model when a new configuration is loaded. A configuration file can only be loaded for the example sheet that has generated the file.
Results

Start the simulation using the SIMULATION->START menu command or the F12 key. The simulation model is compiled and evaluated. The Display Elements show the simulation results on the sheet.

Different panel outputs are visible, depending on the architecture configuration. The output quantities in the control panel are always shown in the sequence: switch_out, lamp_out, power_out, switched_out, and admittance_out.
3.6 Step 6: Powertrain

This example, found in the file system_powertrain.ssh, uses an architecture that models component behavior using a lookup table. The powertrain model reads the lookup table from a file and uses the values for simulation.

The powertrain model provides the speed profile of an engine through a mechanical pin output. This mechanical pin is connected as an input to a Domain-to-Domain (D2D) model that transfers the value of the \textit{ROTATIONAL}\_V pin to the electrical domain. The \texttt{cyc\_udds} architecture is used for the powertrain model to provide a drive cycle.

Some non-electrical parts of a multidomain simulation model can be modeled in the electrical domain. To interface the mechanical domain powertrain model with the electrical switch model, use the Domain-to-Domain model (D2D) from the \textit{Nature Transformations} folder on the \texttt{Tools} tab.

The following figure shows the \texttt{system\_powertrain.ssh} sheet:

![System Powertrain Diagram](image)

Creating the Simulation Model

1. Place and arrange all components shown in the figure above. \textit{Powertrain} and \textit{Resistor} are VHDL-AMS models from the library on the \texttt{Projects} tab. The D2D model is a model from the \textit{Nature Transformations} folder on the \texttt{Tools} tab. (This model does not need to be placed manually: it will be automatically inserted when the powertrain model is connected to the switch model.) The \textit{Ideal Switch} is a model from the \texttt{Switches} folder under the \texttt{Circuit} folder, and the \textit{Pulse Wave} a model from the \texttt{Time Functions} folder under the \texttt{Tools} folder, in the \texttt{basic\_vhdlams} library on the \texttt{AMS} tab. Type CTRL+G to place a ground node.

2. Make the switch control signal pin visible. Double-click the switch symbol to open the Properties dialog, and click the \texttt{Output/Display} tab. Check the \texttt{Pin} box for Control Signal. Click <OK> to apply the changes. The Control Signal pin will be displayed on the component.

3. Connect the models. Place the cursor on a model pin to get the wire cursor and click to enter wire mode. Connect the components as indicated in the figure, setting the beginning, the corners, and the end of a wire with the mouse. Press ESC to end wire mode.
4 Define the powertrain parameters. Double-click the powertrain symbol to open the Properties dialog, and click the «Library» tab. Choose the «cyc_manhattan» architecture from the list. This architecture provides a drive cycle representation of a 1000 second duration. Click <OK> to apply the changes.

5 Define the pulse parameters. Double-click the pulse symbol to open the Properties dialog. Click in the «Value» field, and enter 1 for the amplitude value, 2m for the frequency value, and 1 for the off value. Click <OK> to apply the changes.

6 Define the resistor parameters. Double-click the resistor symbol to open the Properties dialog. Change the model name from res1 to r_load. Click in the «Value» field, and enter 100m for the resistance value. Click <OK> to apply the changes.

7 Place and arrange two 2D Display Elements to display simulation results. The ModelAgent «Displays» tab provides Display Elements to show simulation outputs on the sheet. Drag two 2D View elements onto the sheet.

8 Open the simulation quantities list on one of the 2-D View elements. Double-click the Display Element to open the Properties dialog. Click the Add symbol to open the list of simulation quantities. This process can take some seconds, because all quantities of the model sheet must be scanned.

9 Select the simulation quantities. Check the boxes of output r_load.v and input switch.ctrl. Click <OK> to add these quantities to the list of displayed quantities.

10 Change the display presentation settings. Select each output quantity in the list and change the display presentation settings as necessary. Note the scaling used in the display elements show under “Results” on page 61. Click <OK> to apply the changes. Repeat the steps 8, 9, and 10 for the second Display Element but check the box of powertrain1.vel as output.

---

1 Click the "Add" symbol to open the simulation quantity list.
2 Check the boxes for simulation quantities to display and click <OK>.
3 Select an entry from the list of displayed quantities.
4 Change the display presentation settings in the dialog.
11 Define the simulation parameters. Choose SIMULATION>PARAMETERS, and change the values for Simulation End Time to 1000 sec, Minimum Time Step to 1m sec, and Maximum Time Step to 10m sec. Set Integration Formula to «Trapezoid». Click <OK> to apply the changes.

12 Save the sheet. Choose FILE>SAVE AS, enter file name and directory, and click <OK>.

Overview of Parameter Values

The following table summarizes the parameter values for the models on the sheet:

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Parameter Values</th>
<th>Tab/Library</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>powertrain1</td>
<td>no parameters</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>cyc_manhattan</td>
</tr>
<tr>
<td>D2D1</td>
<td>no parameters</td>
<td>Tools/transformations.smd</td>
<td>SML model</td>
</tr>
<tr>
<td>pulse1</td>
<td>ampl=1, freq[Hz]=2m, off=1; delay[s]=0</td>
<td>AMS/basic_vhdlams.smd</td>
<td>behav</td>
</tr>
<tr>
<td>switch1</td>
<td>ctrl=pulse.val</td>
<td>AMS/basic_vhdlams.smd</td>
<td>behav</td>
</tr>
<tr>
<td>r_load</td>
<td>r_nom [Ω]=100</td>
<td>Projects/vhdlams_tutorial.smd</td>
<td>behav</td>
</tr>
</tbody>
</table>

Powertrain Model

The powertrain model uses the mechanical (ROTATIONAL, V) domain and file I/O functions; consequently, the mechanical_systems and textio package that are defined in the IEEE and STD libraries must be used. The following statements in the model definition include the packages:

```plaintext
LIBRARY IEEE, STD;
USE IEEE.MECHANICAL_SYSTEMS.ALL;
USE STD.TEXTIO.ALL;
```
**Entity Description**

The model provides the characteristic values through a ROTATIONAL_V terminal as indicated in the table below:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Name</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERMINAL</td>
<td>n_out</td>
<td>ROTATIONAL_V</td>
</tr>
</tbody>
</table>

The equivalent VHDL-AMS description for defining the model interface is as follows:

```vhdl-ams
ENTITY powertrain IS
  PORT (
    TERMINAL n_out : ROTATIONAL_V);
END ENTITY powertrain;
```

The following figure shows the equivalent model interface of the powertrain model in the Schematic:

**Architecture Description**

The cyc_manhattan architecture of the powertrain model obtains the speed profile information from the cyc_mph_manhattan_tab.txt file. This file is stored in the project folder.

The following table lists sample values stored in this file:

<table>
<thead>
<tr>
<th>Time [s] *1k</th>
<th>0.013</th>
<th>0.014</th>
<th>0.015</th>
<th>0.016</th>
<th>0.017</th>
<th>0.018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed [mph] *1k</td>
<td>0.0028</td>
<td>0.0049</td>
<td>0.0049</td>
<td>0.0117</td>
<td>0.0134</td>
<td>0.0149</td>
</tr>
</tbody>
</table>

The equivalent VHDL-AMS description for defining the model architecture is as follows:

```vhdl-ams
ARCHITECTURE cyc_manhattan OF powertrain IS
  CONSTANT unit_Time : TIME := 1000 sec;
  FILE cyc : TEXT OPEN READ_MODE IS "cyc_mph_manhattan_tab.txt";
  SIGNAL val : REAL := 0.0;
  SIGNAL ped : TIME := 1 ms;
  QUANTITY vel ACROSS tor THROUGH n_out TO ROTATIONAL_V_REF;
BEGIN
  PROCESS
    VARIABLE buf : LINE;
    VARIABLE t_val_old, t_val_new, t_val, s_val : REAL := 0.0;
  BEGIN
    WHILE NOT ENDFILE(cyc) LOOP
      READLINE(cyc,buf);
      READ (buf,s_val);
      t_val := t_val_new - t_val_old;
      t_val_old := t_val_new;
      ped <= unit_time * t_val;
      val <= s_val*1.0e3;
      WAIT FOR ped;
    END LOOP;
  END PROCESS;
  vel == 8.3776*val'RAMP(1.0,1.0); --vel in rad/sec assuming 2000rpm=>25mph
END ARCHITECTURE cyc_manhattan;
```
A file object cyc of predefined file type TEXT is declared using the following statement:

```plaintext
FILE cyc : TEXT OPEN READ_MODE IS "cyc_mph_manhattan_tab.txt";
```

The WHILE loop reads the file from beginning to end, using the ENDFILE function to test if the file has been read completely. It uses the READLINE procedure to read one line at a time into the buf variable of predefined type LINE. The two values of time and speed from the buf line variable are read into the variables t_val and s_val using the READ procedures. The speed value vel is applied as the across quantity to the output conservative pin, n_out, after converting the value to rotational domain (vel = \(2\pi/60\)). The value vel is calculated assuming that 2000 rpm is equivalent to 25 mph.

The digital SIGNAL value val is transformed to the analog QUANTITY vel with the 'RAMP attribute. The rise time and fall times for the 'RAMP attribute are 1 second each.

To specify data from a different file, for example cyc_mph_udds_tab.txt, a new architecture may be created, as follows:

```plaintext
ARCHITECTURE udds OF powertrain IS
CONSTANT unit_time : TIME := 1000 sec;
FILE cyc : TEXT OPEN READ_MODE IS "cyc_mph_udds_tab.txt";
SIGNAL val : REAL := 0.0;
SIGNAL ped : TIME := 1 ms;
QUANTITY vel ACROSS tor THROUGH n_out TO ROTATIONAL_V_REF;
BEGIN
PROCESS
VARIABLE buf : LINE;
VARIABLE t_val_old, t_val_new, t_val, s_val : REAL := 0.0;
BEGIN
WHILE NOT ENDFILE(cyc) LOOP
READLINE(cyc,buf);
READ (buf,t_val_new);
READ (buf,s_val);
t_val := t_val_new - t_val_old;
t_val_old := t_val_new;
ped <= unit_time * t_val;
val <= s_val*1.0e3;
WAIT FOR ped;
END LOOP;
WAIT;
END PROCESS;
vel := 8.3776*val*RAMP(1.0,1.0); --vel in rad/sec assuming 2000rpm=>25mph
END ARCHITECTURE udds;
```

The only difference between the two architectures is the file name specified. However, the multiple architectures allow the sheet to be simulated using different speed profiles by simply changing the architecture, enabling multiple tests to be quickly performed on the powernet system.
**Results**

Start the simulation using the SIMULATION>START menu command or the F12 key. The simulation model is compiled and evaluated. The Display Elements show the simulation results on the sheet.

The powertrain model provides the values for simulation based on the values in the characteristic file. Whenever the electrical switch is turned on, the speed output from the powertrain model is transferred to the load resistor.
4 Case Studies

The case study examples in this chapter extend the basic set of features introduced so far. They include analog, mixed-signal, and mixed-technology models.

The first case explains different modeling methods using the modified battery model from the Automotive Powernet System example. The second case uses a fuse model to illustrate the use of the VHDL-AMS Model Wizard to create a model and provide symbol animation. The third case shows the influence of model descriptions on simulation time using a detailed and an averaged claw-pole alternator. The fourth case uses a load model to explain the concept of multiple architectures for a single model, and the concept of component instantiation as a way to reuse a particular model behavior. The fifth case uses a linear drive system and a simple solenoid model to explain multidomain modeling. The sixth case uses a PWM controller to explain mixed-signal modeling, and an automotive alarm system to illustrate using a stimulus generator, exporting VHDL-AMS models, and using foreign models.

Each of the case studies contains:

- A Concepts section that describes the model design concept illustrated
- One or more Background sections that describe the physical systems being modeled
- One or more Model sections that describe the methods used to define the models

This chapter contains information on:

- Multiple modeling styles: battery model
- Automated model development using the VHDL-AMS Wizard: fuse model
- Detailed and averaged model development: claw-pole alternator model
- Multilevel modeling techniques: load model
- Multidomain system modeling: linear drive system, solenoid
- Mixed signal modeling: DC-DC converter with PWM, automotive alarm system

Using Example Sheets

The examples described in this chapter and the required library vhdlams_tutorial.smd are available on the Student Version CD. SIMPLORER 7.0 is needed to create and simulate the examples. The Student Version has some limitations in the model design. If an example cannot be executed with the student version, there will be a note on its sheet. Since the Student Version limits the size of a design, some models need to be excluded from the simulation. To exclude models, select the models and choose ELEMENT>DONT ADD TO MODEL DESCRIPTION. The selected models appear hatched, indicating that they are not included in the simulation.

The examples can be loaded from the tutorials files, or can be created from scratch. Open the case_study_examples.ssc project file in the Tutorial Examples folder to access all sheets referred to in this chapter. The examples can be done sequentially or started with any example, since each case explains a different concept and is not based on a previous case. If there are no comments about changing parameter values in these examples, the default values are used.
VHDL-AMS Modeling Features

The following table lists VHDL-AMS features and models used in the case studies (including the example file name), and indicates if the examples can be run in the Student Version (SV).

<table>
<thead>
<tr>
<th>Case</th>
<th>VHDL-AMS Features</th>
<th>Model/Sheet Name</th>
<th>SV</th>
</tr>
</thead>
</table>
| 1    | • Graphical, structural, behavioral modeling methods  
      • Component instantiation  
      • Graphical and text subsheets | Battery  
      case_study_battery.ssh | yes |
| 2    | • Automated model development  
      • VHDL-AMS Model Wizard  
      • Component instantiation  
      • Symbol animation | Fuse  
      case_study_fuse_lamps.ssh | partly |
| 3    | • Modeling descriptions to reduce simulation time  
      • Graphical and text subsheets | Averaged and detailed claw-pole alternator  
      case_study_clawpole_avg.ssh  
      case_study_clawpole_math.ssh | no |
| 4    | • Multiple architectures  
      • Component instantiation  
      • Symbol animation | Load models  
      case_study_loads.ssh | partly |
| 5    | • Multidomain modeling  
      • Maxwell coupling | Linear Drive and Solenoid  
      case_study_em_linear_drive.ssh  
      case_study_em_solenoid.ssh | partly |
| 6    | • Mixed-signal modeling  
      • Stimulus generator  
      • Multidomain sensor modeling  
      • Foreign models  
      • Export of VHDL-AMS netlists | PWM Controller and Automotive Alarm System  
      case_study_pwm_dcdc.ssh  
      case_study_automotive_alarm_system.ssh | yes |
4.1 Different Modeling Styles: Battery

Concepts

The first case study uses a battery model to illustrate how different modeling styles can be used in SIMPLORER to obtain the same behavior.

The battery is modeled in three different ways:

- Graphical description using subsheet modeling
- Structural description in ASCII text using component instantiation
- Behavioral description in ASCII text using differential algebraic equations

Each method provides the same results but has advantages and disadvantages. Although the three modeling styles provide the same output, there are some fundamental differences between the modeling styles.

**Graphical Modeling Style**

The graphical modeling style is the easiest method for developing the battery model, since all the necessary submodels are available in the library. This method does not require knowledge of VHDL-AMS syntax and programming, and the models can be dragged and dropped from a library onto the sheet directly. This modeling method is recommended for beginners.

- **ADVANTAGE:** This modeling method is easy to use.
- **DISADVANTAGE:** The building block models must be obtained from somewhere.

**Structural Modeling Style**

The structural modeling style is useful when a design needs to be developed using specific models in VHDL-AMS text format. When this style is used, the end result is a VHDL-AMS text netlist.

- **ADVANTAGE:** This method is good for users who are more comfortable in writing VHDL-AMS models in ASCII format.
- **DISADVANTAGE:** The building block models and advanced knowledge of AMS syntax are required.

**Behavioral Modeling Style**

The behavioral modeling style is useful for describing model function in an abstract manner, using equations and programming constructs. This modeling method is recommended for experts.

- **ADVANTAGE:** The modeling and simulation can be optimized because the model description is self-contained.
- **DISADVANTAGE:** The model description may not correspond to any real physical model.
The following figure shows the basic test circuit for all of the modeling styles. In each style, the battery is replaced with an equivalent model. The current source has a value of –1A, indicating that it draws current from the battery model.

The battery model is designed as a simple circuit with two resistors and two capacitors as shown in the following circuit:

The battery model also accepts two parameters: \( v_{\text{init}} \) that specifies the initial voltage of the battery, and \( \text{factor} \), a parameter that expresses the capacity factor and aids in simple scaling of the battery. The internal resistance \((r_i=10\,\text{m})\), diffusion resistance \((r_d=40\,\text{m})\), the fast capacitor \((c_f=60\times\text{factor})\), and the slow capacitor \((c_s=20000\times\text{factor})\) have constant values.

The model calculates the circuit based on the following equations:

\[
\begin{align*}
    v_{\text{ri}} &= i_{\text{ri}} \times r_i \\
    \frac{d(v_{fc})}{dt} &= \frac{1}{(fc \times \text{factor})} \times i_{fc} \\
    v_{rd} &= i_{rd} \times r_d \\
    \frac{d(v_{sc})}{dt} &= \frac{1}{(sc \times \text{factor})} \times i_{sc}
\end{align*}
\]

The \( \text{factor} \) value is used to scale the two capacitances from their default value. Thus, \( \text{factor} \) is a normalized charge capacity for the battery.

See also “Battery Model” on page 36. Unlike the battery model in the Automotive Powernet System example, this battery model does not provide State of Charge (SoC) as an output parameter.
Model Style 1: Graphical Description

This modeling style uses models from the library to graphically generate the battery model. To create the graphical battery subsheet, do the following:

1. Choose SHEET>SUBSHEET>NEW GRAPHICAL. The mouse pointer becomes a cross wire.
2. Drag the subsheet symbol to the desired size by holding the left mouse button. A rectangular symbol is displayed, and an empty sheet opens.

3. Place and arrange all models used corresponding to the following circuit:

   ![Battery Circuit Diagram]

   *Resistor and Capacitor are VHDL-AMS models from the library on the «Projects» tab. The Voltmeter is a model from the Measurement folder of the basic_vhdlams library on the «AMS» tab.*

4. Create the model Interface:
   - Establish conservative nodes (terminal p and n) using SHEET>SUBSHEET>CREATE PIN/PARAMETERS. Conservative nodes are connectors from the substructure to the next higher model level. Define the name of the conservative node, click <OK>, and select ELECTRICAL as the nature type. The new pin symbol can now be connected with the wiring.
   - Establish non-conservative nodes (factor, v_init, v_out) using SHEET>SUBSHEET>CREATE PIN/PARAMETERS. Non-conservative nodes are used to change parameters. Non-conservative nodes need to be specified as input, output, or input and output. In this example factor and v_init are input parameters, and v_out is an output parameter.

5. A pin symbol appears for each non-conservative node, even when the pin is not connected through a wire to a model parameter.

   ![Pin Symbols]

   Switch to the next higher model level using «Level Up» in the subsheet shortcut menu. Double-click the subsheet symbol to open the Properties dialog. In the «Output/Display» dialog the defined terminals are listed.

   To view the v_out value through a Probe Element on the sheet, click with the right mouse button on the subsheet symbol, and select «Probe». A Probe Element with v_out as output quantity is added to the symbol. Change the presentation format by clicking on the Probe Element with the right mouse button and select «Show as Graphic».
Model Style 2: Structural Description

This modeling style uses the structure of the battery model developed with Style 1 to describe the battery model in a text format. In this example VHDL-AMS Style are used to instantiate the required components, similar to Style 1. This differs from Method 1, however, in that the netlist is specified using VHDL-AMS text, whereas it was generated from graphical specifications in Method 1.

For the battery model, we require two capacitor models and two resistor models that are available in the library. The models can be accessed from the library if the following statements are added at the beginning of the entity description:

```
LIBRARY WORK;
USE WORK.ALL;
```

The `USE` statement specifies that all models and packages that are defined within the `WORK` library are now accessible from any architecture of the battery model.

**Entity Description**

The entity description of the structural battery model uses static parameters (constant value inputs evaluated only at the beginning of the simulation) and terminals in the interface. The battery model has two parameters for the factor and initial voltage (`factor`, `v_init`) as defined in the `GENERIC` declaration. These parameters are of type `REAL` with specific default values. The battery provides its electrical output through a pair of electrical terminals and provides the voltage across the battery through the `v_out` output defined in the `QUANTITY` statement.

```
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
LIBRARY WORK;
USE WORK.ALL;
ENTITY bat_multi IS
  GENERIC:
    factor : REAL := 1.0;
    v_init : VOLTAGE := 12.0;
  PORT:
    TERMINAL p, m : ELECTRICAL;
    QUANTITY v_out : OUT VOLTAGE := 0.0;
END ENTITY bat_multi;
```

**Architecture Description**

The `struct` architecture of the battery model defines two internal terminals of type `ELECTRICAL`, `t1` and `t2`, within the model of type `ELECTRICAL`. Also, the model defines constant resistance values for `rd` and `ri`, as well as constant capacitance values for `fc` and `sc`.
The following model instantiation uses the model `cap` from the `WORK` library. The **GENERIC MAP** and **PORT MAP** statements assign the model parameters and terminals to the `cap` entity names, `c_nom`, `v_init`, `p`, and `m`.

```vhdl
fc1: ENTITY cap(behav)
  GENERIC MAP (c_nom => fc*factor, v_init => v_init)
  PORT MAP (p => t1, m => m);
END ENTITY;
```

The equivalent VHDL-AMS description for defining the model architecture is as follows:

```vhdl
ARCHITECTURE struct OF bat_multi IS
  CONSTANT ri: RESISTANCE := 1.0e-2;
  CONSTANT fc: CAPACITANCE := 60.0;
  CONSTANT rd: RESISTANCE := 4.0e-2;
  CONSTANT sc: CAPACITANCE := 2.0e4;
  TERMINAL t1,t3 : ELECTRICAL;
  QUANTITY v ACROSS p TO m;
BEGIN
  fc1: ENTITY cap(behav)
    GENERIC MAP (c_nom => fc*factor, v_init => v_init)
    PORT MAP (p => t1, m => m);
  sc1: ENTITY cap(behav)
    GENERIC MAP (c_nom => sc*factor, v_init => v_init)
    PORT MAP (p => t2, m => m);
  ri1: ENTITY res(behav)
    GENERIC MAP (r_nom => ri)
    PORT MAP (p => p, m => t1);
  rd1 : ENTITY res(behav)
    GENERIC MAP (r_nom => rd)
    PORT MAP (p => t1, m => t2);
  v_out == v;
END ARCHITECTURE struct;
```

**Model Style 3: Behavioral Description**

This modeling style uses differential algebraic equations (DAE) to describe the behavior of the battery model. It does not instantiate other components, so the resulting model is more compact and consequently, more efficient.

**Entity Description**

The entity description of the behavioral battery model uses static parameters (constant value inputs evaluated only at the beginning of the simulation) and terminals in the interface. The battery model has two parameters for the factor and initial voltage (`factor`, `v_init`) as defined in the **GENERIC** declaration. These parameters are of type REAL with specific default values.

```vhdl
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
LIBRARY WORK;
USE WORK.ALL;
ENTITY bat_multi IS
  GENERIC(
    factor : REAL := 1.0;
    v_init : VOLTAGE := 12.0);
  PORT(
    p,m : ELECTRICAL);
  QUANTITY v_out : OUT VOLTAGE := 0.0);
END ENTITY bat_multi;
```
Architecture Description

The equivalent VHDL-AMS description for defining the model architecture is as follows:

```vhdl
ARCHITECTURE behav OF bat_multi IS
  TERMINAL t1, t2: ELECTRICAL;
  QUANTITY v_ri ACROSS i_ri THROUGH p TO t1;
  QUANTITY v_fc ACROSS i_fc THROUGH t1 TO m;
  QUANTITY v_rd ACROSS i_rd THROUGH t1 TO t2;
  QUANTITY v_sc ACROSS i_sc THROUGH t2 TO m;
  QUANTITY v ACROSS p TO m;
  CONSTANT ri: RESISTANCE := 1.0e-2;
  CONSTANT fc: CAPACITANCE := 60.0;
  CONSTANT rd: RESISTANCE := 4.0e-2;
  CONSTANT sc: CAPACITANCE := 2.0e4;
BEGIN
  BREAK v_fc => v_init, v_sc => v_init;
  v_ri == i_ri * ri;
  v_fc'DOT == 1.0/(fc*factor) * i_fc;
  v_rd == i_rd * rd;
  v_sc'DOT == 1.0/(sc*factor) * i_sc;
  v_out == v;
END ARCHITECTURE behav;
```

Two internal electrical terminals, \( t1 \) and \( t2 \), are created as in Style 2. The four quantity statements define through and across quantities for the two resistors and capacitors. Together, these five statements define the internal topology of the model.

After the BREAK statement, the four equations describe the branch characteristics of each internal resistor and capacitor. To avoid discontinuities during simulation, the BREAK statement is used to initialize the voltages across the fast and slow capacitors with the value of \( v_{\text{init}} \) (initial voltage).

The voltage between the terminals is provided as output voltage, with the simultaneous statement \( v_{\text{out}} = v \).

Results

The Display Element shows the battery output voltage of the simulation. Since the battery model is connected to a constant current sink, the battery voltage decreases with time as the battery slowly discharged. A fast discharge, governed by capacitor \( fc \), is followed by a slow discharge, governed by capacitor \( sc \).

Even though different modeling styles have been adopted, the outputs of all three battery models are identical, indicating that all models display the same behavior. To show all three curves distinctly, the output voltages of the behavioral and structural battery models are offset in the graphic.
4.2 Automated Model Development Using VHDL-AMS Wizard: Fuse

Concepts

This case study illustrates the use of the VHDL-AMS Model Wizard to develop a model, the creation of an animated symbol for the model, and the use of the model in a simulation.

Using the VHDL-AMS Wizard

Fuse Model Equivalent

The following is the schematic equivalent of the VHDL-AMS text netlist that will be created using the VHDL-AMS Wizard. The fuse model is an example of a multidomain, mixed-signal model involving the electrical and thermal domains. The fuse model can be divided into three parts: a thermal network, an electrical network, and a digital logic unit.

The thermal network models the self-heating behavior of the fuse. It also considers the fuse's dynamic heat exchange with the surroundings using a thermal resistance and a thermal capacitance. The influence of the ambient temperature on the fuse is modeled by a temperature source connected to the thermal network. The digital logic determines the threshold crossing of the fuse temperature over the specified maximum temperature. The output from the digital logic is used to control a switch in the electrical network that switches between the fuse resistance and an off resistance. The fuse resistance is calculated based on the temperature of the thermal network and the material-specific parameters, while the off resistance is provided as a parameter for the model.
Create a VHDL-AMS Framework for the Fuse Model

1. In the SSC Commander window, click on the VHDL-AMS Wizard icon.
2. Click on the FILE>NEW menu option or the New document icon. If a project is already open, then this step can be ignored.

3. Enter `fuse` as the entity name. Note that the architecture name is automatically filled in.
4. In the «Use Libraries» frame, select the `ieee` library and the `basic_vhdlams` library.
5. In the «Use Packages» frame, select the `ieee.electrical_systems` and `ieee.thermal_systems` packages. Do not select any models in the «Use Models» frame.

If the full names of the packages are obscured, drag the bottom right corner of the dialog to resize the dialog and expand the areas that list the packages.
6. Click <OK>. An empty VHDL-AMS code framework is created and the equivalent hierarchical representation of the model is provided in the model tree in the left-hand pane.

7. Save the model file as `fuse.vhd`.

**Describe the Entity**

1. Open (expand) the model tree under Fuse.
2. Open the Entity folder under Fuse in the model tree. Right-click on the Generic or Port folders in the Model Tree and select «Edit Entity», or choose `MODULE>EDIT ENTITY>FUSE`. The Edit Entity dialog appears.
3. In the «Ports» frame, click five times on the Add Port button to create five ports (two terminal ports, two quantity ports and one signal port). Configure the ports as shown:

   ![Port Configuration Table]

   Port input quantities and signals can have an initial value. To set an initial value, enter it in the «Value» field of that item in the dialog box.
4 In the «Generic» frame, click seven times on the Add Generic button to create seven generic value definitions. Configure them as shown below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>th_res</td>
<td>thermal_resistance</td>
<td>500</td>
<td>Thermal Resistance</td>
</tr>
<tr>
<td>th_cap</td>
<td>thermal_capacitance</td>
<td>1.35</td>
<td>Thermal Capacitance</td>
</tr>
<tr>
<td>temp_max</td>
<td>temperature</td>
<td>673</td>
<td>Maximum Temperature</td>
</tr>
<tr>
<td>temp_ref</td>
<td>temperature</td>
<td>220</td>
<td>Reference Temperature</td>
</tr>
<tr>
<td>r_eff</td>
<td>resistance</td>
<td>10</td>
<td>(1) Resistance</td>
</tr>
<tr>
<td>r</td>
<td>resistance</td>
<td>1.8m</td>
<td>Resistance at Ambient Temperature</td>
</tr>
<tr>
<td>alpha</td>
<td>soil</td>
<td>3.23%</td>
<td>Temperature Coefficient</td>
</tr>
</tbody>
</table>

5 Click <OK>. The VHDL-AMS code for the entity declaration is automatically generated in the editor window.

The VHDL-AMS Wizard allows dual editing of the models from the editor window as well as the dialogs. Changes made in either editing mode will be synchronized in both panes when the model is scanned again by using the Check button.
Describe the Architecture

1. Right-click on the arch_fuse folder under the Architecture folder in the model tree and use the short cut menu to choose «Edit Architecture Declaration». Add the following ten declarations and one branch quantity and click <OK>:

2. Right-click on the arch_fuse folder and choose «Instantiate Component».

3. From the basic_vhdlams library model tree, double-click on the following models to add them to the fuse model:

<table>
<thead>
<tr>
<th>Electrical Network</th>
<th>Thermal Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit &gt; Passive Elements &gt; Resistor (double-click twice for two models)</td>
<td>Physical Domains &gt; Thermal &gt; Temperature Source</td>
</tr>
<tr>
<td>Circuit &gt; Switches &gt; Ideal Transfer Switch</td>
<td>Physical Domains &gt; Thermal &gt; Heat Flow Source</td>
</tr>
<tr>
<td>Measurement &gt; Electrical &gt; Ammeter</td>
<td>Physical Domains &gt; Thermal &gt; Thermal Resistance</td>
</tr>
<tr>
<td></td>
<td>Physical Domains &gt; Thermal &gt; Thermal Capacitance</td>
</tr>
<tr>
<td></td>
<td>Measurement &gt; Thermal &gt; Thermometer</td>
</tr>
</tbody>
</table>
4 In the Instantiate Component dialog, click on each of the models and set the entity names and port map values for each model as shown in the following table. Note that most values can simply be selected from a drop-down menu. Only two equations need to be entered manually.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Name</th>
<th>Port Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>fuse_res</td>
<td>r: 10 \times (1.0 + \alpha \times (\text{temp_int} - \text{temp_ref}))</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>terminal_e3</td>
</tr>
<tr>
<td>m</td>
<td>blown_res</td>
<td>r: \text{off_res}</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>terminal_e2</td>
</tr>
<tr>
<td>m</td>
<td>three_port_sw</td>
<td>ctrl: switch Ctrl qty</td>
</tr>
<tr>
<td></td>
<td>n1</td>
<td>plus</td>
</tr>
<tr>
<td></td>
<td>n2</td>
<td>terminal_e2</td>
</tr>
<tr>
<td></td>
<td>n3</td>
<td>terminal_e1</td>
</tr>
<tr>
<td>am</td>
<td>fuse_res_amm</td>
<td>i: fuse_res_i</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>terminal_e1</td>
</tr>
<tr>
<td></td>
<td>m</td>
<td>terminal_e3</td>
</tr>
</tbody>
</table>
Right-click on the arch_fuse folder and choose «Edit Architecture Body». Insert the following text at the bottom of the existing code.

5 The fuse model has been successfully created. The next step is to include it in a schematic and animate the model symbol. Save the VHDL-AMS file and exit the VHDL-AMS Wizard.

**Import the Model into Schematic**

If the case studies project is not open, open the case_study_examples.ssc project and create a new schematic in the project. In Schematic, choose SHEET>SUBSHEET>New VHDL-AMS and use the click and drag mouse action to create a new VHDL-AMS Text Subsheet. Within the subsheet, choose «Import» from the shortcut menu to browse to the new fuse model and import it into the subsheet. Select «Level Up» from the shortcut menu to return to the sheet.

**Animating the Model Symbol**

The method described here is a shortcut method that uses an existing symbol definition to animate a symbol. Typically, a symbol animation will be developed graphically in the Symbol...
Case Studies

Editor. See “Symbol Animation” on page 101 for more information on creating symbol animation graphics.

An animated symbol for the fuse model can be created with the contents of the fuse_symbol.txt file. Open the fuse_symbol.txt file in a text editor. In Schematic, right click on the fuse model and select «Edit Symbol» to open the Symbol Editor. In the Symbol Editor, choose EDIT>SYMBOL TEXT and replace the default text in the symbol text dialog with the text from fuse_symbol.txt.

Click <OK> and Exit the Symbol Editor. The animated symbol is now ready for use in the Schematic.

See “Symbol Animation” on page 101 to learn how to graphically create the symbol animation for a model.

Background: Fuse for Lamps in an Automotive Subsystem

The example below demonstrates the use of the fuse model in conjunction with several lamps in an automotive subsystem. The subsystem is powered by the SoC battery model and has five lamp models. The lamp models are switched on sequentially using a digital control text sub-sheet model in VHDL-AMS. The sequential activation of the individual lamp models draws additional current from the battery model. The activation of the fifth lamp model blows the fuse and disconnects the lamps from the power system. For this example to run in the student version of SIMPLORER, the bat_soc1, lamp1, lamp2, and lamp3 models must be deactivated; e1 and its associated ground pin, and r1 must be activated.
Model: Digital Control Model

Entity Description

The digital control model has five output control signals of type REAL that will be used to switch each of the five lamps on and off. The equivalent VHDL-AMS description for the model interface is as follows:

```
ENTITY control IS
  PORT(
    SIGNAL ctrl1 : OUT REAL := 0.0; --Control Signal 1
    SIGNAL ctrl2 : OUT REAL := 0.0; --Control Signal 2
    SIGNAL ctrl3 : OUT REAL := 0.0; --Control Signal 3
    SIGNAL ctrl4 : OUT REAL := 0.0; --Control Signal 4
    SIGNAL ctrl5 : OUT REAL := 0.0; --Control Signal 5
  );
END ENTITY control;
```

Architecture Description

The model architecture describes the signal characteristics of the digital control model. It turns on each of the control signals at 10 second intervals using concurrent signal assignment statements. The signal assignments are delayed through the use of the `AFTER` keyword.

```
ARCHITECTURE arch_control OF control IS
BEGIN
  ctrl1 <= 1.0 AFTER 1ms;
  ctrl2 <= 1.0 AFTER 10ms;
  ctrl3 <= 1.0 AFTER 20ms;
  ctrl4 <= 1.0 AFTER 30ms;
  ctrl5 <= 1.0 AFTER 40ms;
END ARCHITECTURE admittance;
```

Model Parameters

The lamp models are parameterized to have a nominal power consumption of 60 Watts. The fuse model has the following parameters:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Used Unit</th>
<th>Default</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fuse</td>
<td>50</td>
<td>K/A</td>
<td>50.0</td>
<td>K/A</td>
<td></td>
</tr>
<tr>
<td>fuse_cap</td>
<td>10m</td>
<td>J/K</td>
<td>3.19</td>
<td>J/K</td>
<td></td>
</tr>
<tr>
<td>fuse_res</td>
<td>673</td>
<td>K</td>
<td>673.0</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>fuse_tmax</td>
<td>250</td>
<td>K</td>
<td>250.0</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>fuse_x</td>
<td>15</td>
<td>Dm</td>
<td>1.00E03KOHm</td>
<td>Dm</td>
<td></td>
</tr>
<tr>
<td>fuse</td>
<td>10m</td>
<td>Dm</td>
<td>0.0181</td>
<td>Dm</td>
<td></td>
</tr>
<tr>
<td>fuse</td>
<td>5.0m</td>
<td>m</td>
<td>0.0092</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>fuse</td>
<td>250</td>
<td>K</td>
<td>0.0</td>
<td>K</td>
<td></td>
</tr>
</tbody>
</table>
```

Most of the parameters of the fuse model are material-specific (such as temperature coefficient of resistance (alpha), maximum temperature (temp_max), resistivity (rho), and so on). Fuse model datasheets provide information about the fuse materials and blow-time characteristics. This data can be used with multirun simulations to determine appropriate values for thermal resistance and thermal capacitance.

Simulation Parameters

Choose `SIMULATION>PARAMETERS`, and change the default value for Simulation End Time to 50 sec, Minimum Time Step to 100u sec, and Maximum time step to 10m sec. Click `<OK>` to apply the changes.
The first Display Element illustrates the variation of the fuse temperature with time. As each lamp is turned on, the fuse temperature is found to increase and the switching on of the fifth lamp blows the fuse at approximately 40 sec.

The second Display Element shows the fuse state value as an output. The third Display Element shows the digital control outputs and illustrates the activation curves of the lamps. The fourth Display Element shows the State of Charge curve of the battery and illustrates the discharging of the battery as each lamp load is activated, and the “leveling off” after the fuse blows.
4.3 Detailed and Average Model Development: Claw-Pole Alternator

Concepts

This case study illustrates the modeling of detailed and averaged claw-pole alternator models, as discussed in Vahe Caliskan: *Modeling and Simulation of a Claw-Pole Alternator, Detailed and Averaged Models* [3].

This example emphasizes a key design choice to be made when simulating large circuits. A complicated three phase model with rectifier can be used to maintain accuracy, but at the expense of long simulation times. A simple averaged model can be used to decrease simulation time, with some sacrifice in accuracy. Both design methodologies, as presented in the publication, are described in this chapter.

A detailed model is used when the processes are highly transient, with spike currents, switching, and so on. The detailed claw-pole model contains a rectifier bridge with six diodes to model the switching characteristics accurately. An averaged model is used to model energy exchange over extended periods of time, such as load balance simulations. The averaged claw-pole model contains simplified models of the armature and rectifier.

The two alternator design circuits are developed as graphical subsheet models, using VHDL-AMS library models and textual subsheets. The models are used in conjunction with the SOC Battery model and the Powertrain model.

Background: Detailed Model of a Claw-Pole Alternator

The detailed alternator model is separated into three modules: a three phase generator, a three-phase rectifier, and a switching voltage regulator. The following figure shows the
The three-phase rectifier model uses diodes from the Circuit folder of the basic_vhdlams library on the «AMS» tab. The control circuit uses block models from the Blocks folder in the same library. The Claw-Pole Alternator model is available from the library on the «Project» tab.
Model: Mathematical Claw-Pole Model

The parameters and terminals for the mathematical model are as follows:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Name</th>
<th>Property</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC</td>
<td>rs</td>
<td>RESISTANCE</td>
<td>33m</td>
<td>Stator Winding Resistance [Ω]</td>
</tr>
<tr>
<td></td>
<td>lls</td>
<td>INDUCTANCE</td>
<td>18u</td>
<td>Stator Leakage Inductance [H]</td>
</tr>
<tr>
<td></td>
<td>lms</td>
<td>INDUCTANCE</td>
<td>180u</td>
<td>Stator Magnetizing Inductance [H]</td>
</tr>
<tr>
<td></td>
<td>rr</td>
<td>RESISTANCE</td>
<td>1.44</td>
<td>Rotor Winding Resistance [Ω]</td>
</tr>
<tr>
<td></td>
<td>llr</td>
<td>INDUCTANCE</td>
<td>200m</td>
<td>Rotor Leakage Inductance [H]</td>
</tr>
<tr>
<td></td>
<td>lmr</td>
<td>INDUCTANCE</td>
<td>300m</td>
<td>Rotor Magnetizing Inductance [H]</td>
</tr>
<tr>
<td>p</td>
<td>REAL</td>
<td></td>
<td>12</td>
<td>Number of Poles</td>
</tr>
<tr>
<td>theta0</td>
<td>ANGLE</td>
<td></td>
<td>0</td>
<td>Initial Rotor Angle [rad]</td>
</tr>
<tr>
<td>k</td>
<td>REAL</td>
<td></td>
<td>1</td>
<td>Coeff. Mag. Coupling Stator-Rotor</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>a, b, c</td>
<td>ELECTRICAL</td>
<td></td>
<td>Phase a, b, c voltage of stator</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>ELECTRICAL</td>
<td></td>
<td>Neutral terminal of alternator</td>
</tr>
<tr>
<td></td>
<td>fp, fm</td>
<td>ELECTRICAL</td>
<td></td>
<td>Field positive/negative terminal connection</td>
</tr>
<tr>
<td></td>
<td>shaft</td>
<td>ROTATIONAL_V</td>
<td></td>
<td>Rotor mechanical angular velocity</td>
</tr>
</tbody>
</table>

The equations provided for the generator model are as follows:

\[
\frac{d\lambda_a(t)}{dt} = v_a(t) - R_s i_a(t)
\]

\[
\frac{d\lambda_b(t)}{dt} = v_b(t) - R_s i_b(t)
\]

\[
\frac{d\lambda_c(t)}{dt} = v_c(t) - R_s i_c(t)
\]

\[
\frac{d\theta_r}{dt} = \omega_r \quad \theta_c = \left(\frac{p}{2}\right)\theta_e
\]

These equations describe an ideal three-phase generator, with windings distributed to generate perfectly sinusoidal voltages. But a real claw-pole alternator will generate voltages with significant distortion. This effect can be accounted for in finite element models, or by incorporating high order harmonics in the inductance terms.

See also Bai, H. and et. al.: *Analytical Derivation of a Coupled-Circuit Model of a Claw-Pole Alternator with Concentrated Stator Winding* [5].
M is the mutual inductance between stator coils and rotor, $\theta_r$ is the rotor angle, $\theta_e$ is the electrical angle, $L_s$ is the stator self-inductance, and $L_r$ is the stator self-inductance.

$$L_s = 11_s + l_{ms}$$
$$L_r = 11_r + l_{mr}$$
$$M = k \cdot \sqrt{l_{ms} \cdot l_{mr}}$$

The rectifier and field diodes all require RC snubbers to properly function during simulation. The values are set differently, according to the stator and rotor inductances. The voltage regulator will switch the field ON and OFF at approximately 140 Hz, so the snubber C is chosen for resonance at 1400 Hz. Then the snubber R is chosen for critical clamping.

**Stator:**

$$L_s = 11_s + l_{ms} = 18 \mu H + 180 \mu H = 198 \mu H$$

$$C_{sns} = \frac{1}{L_s \omega^2} = \frac{1}{198 \mu H \cdot (2 \cdot \pi \cdot 1400 Hz)^2} = 65 \mu F$$

$$R_{sns} = \frac{L_s}{C_{sns}} = \frac{198}{65} = 1.7 \Omega$$

**Rotor:**

$$L_r = 11_r + l_{mr} = 200 mH + 300 mH = 500 mH$$

$$C_{snr} = \frac{1}{L_r \omega^2} = \frac{1}{500 mH \cdot (2 \cdot \pi \cdot 1400 Hz)^2} = 26 nF$$

$$R_{snr} = \frac{L_r}{C_{snr}} = \frac{500m}{26n} = 4.4 k\Omega$$

The detailed model of the claw-pole can be developed as a VHDL-AMS model with the following description:

```vhdl
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MECHANICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;
ENTITY cp_math IS
  GENERIC(
    rs : RESISTANCE := 33.0e-3; -- Stator winding resistance
    ll_s : INDUCTANCE := 18.0e-6; -- Stator leakage inductance
    lms : INDUCTANCE := 180.0e-6; -- Stator magnetizing inductance
    rr : RESISTANCE := 3.44; -- Rotor winding resistance
    ll_r : INDUCTANCE := 200.0e-3; -- Rotor leakage inductance
    lmr : INDUCTANCE := 300.0e-3; -- Rotor magnetizing inductance
    p : REAL := 12.0; -- Number of poles
    theta0 : ANGLE := 0.0; -- Initial rotor angle
    k : REAL := 1.0; -- Coefficient of magnetic coupling stator-rotor
  );
  PORT(
    TERMINAL a, b, c : ELECTRICAL;
    TERMINAL n, fp, fm : ELECTRICAL;
    TERMINAL shaft : ROTATIONAL_V);
END ENTITY cp_math;
```
First, the flux linkage is defined in terms of time-varying inductances and currents. Then, the four coil voltage equations are written using Faraday’s Law.

The outputs include coil flux linkage, phase inductances, rotor angle, and electrical angle. **CONSTANT** definitions are used to pre-process the inductance parameters.

This detailed model of the claw-pole requires small time steps and a long simulation time to accurately simulate the switching characteristics (HMIN=50µs, HMAX=5ms, TEND=4.5s).
The first Display Element shows the three-phase output voltages from the mathematical claw-pole model. The Powertrain model's velocity output is plotted in the second graph. The voltage and frequency increase linearly with the powertrain velocity. The field switching curve is displayed as *cpgen.df*, and the battery output is displayed in the fourth graph.
Background: Averaged Model of Claw-Pole Alternator

In this averaged model, the three rectified phases are simplified to an equivalent DC machine. The green portion is the equivalent armature, representing the speed voltage generated in the stator. The blue portion contains the field circuit, and the orange portion contains the voltage regulator. The yellow portion represents the rectifier voltage drop, and prevents motor operation of the alternator. This model is better suited for systems where transient characteristics are not important.

The following figure shows the design of the averaged alternator model:

Model

Speed-Voltage Model

The averaged claw-pole alternator uses a speed-voltage model that is modeled as a current source. It provides a terminal voltage value that is proportional to the alternator speed and field current, according to the equation:

$$V = k \cdot n \cdot i$$

This is the same equation describing a separately excited DC machine, where $i$ is the field current.
The speed-voltage model is developed as a VHDL-AMS text subsheet. The entity and architecture description are as follows:

```vhdl-ams
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;
ENTITY speedvolt IS
  GENERIC( k : REAL := 1.0);
  PORT( QUANTITY n_val : REAL := 0.0;
        QUANTITY ifield_val : CURRENT := 0.0;
        TERMINAL pos, neg : ELECTRICAL);
END ENTITY speedvolt;
ARCHITECTURE behav OF speedvolt IS
  QUANTITY v ACROSS i THROUGH pos TO neg;
BEGIN
  v == k*n_val*ifield_val;
END ARCHITECTURE behav;
```

**Stator-Impedance Model**

The stator impedance in the averaged model of the claw-pole alternator uses a speed-dependent resistive drop given by the following equation:

\[
Z(n) = \sqrt{R_s^2 + \left(\frac{\pi}{30} \cdot \frac{p}{2} \cdot n \cdot L_s\right)^2}
\]

The model accepts the stator resistance, stator inductance, number of poles, and alternator speed as inputs. It provides a voltage/current output according to the impedance equation. \(Z(n)\) is the magnitude of the equivalent stator impedance. The inductive part of this impedance increases with frequency, or equivalently, with rotor speed \(n\). When \(R_s\) is greater than zero, \(Z(n)\) is always greater than zero.

The stator-impedance is also modeled as a VHDL-AMS text subsheet. The entity and architecture description are as follows:

```vhdl-ams
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.ALL;
ENTITY statorz IS
  GENERIC( rs : RESISTANCE := 33.0e-3;
           ls : INDUCTANCE := 177.0e-6;
           p : REAL := 12.0);
  PORT( QUANTITY n_val : REAL := 0.0;
        TERMINAL pos, neg : ELECTRICAL);
END ENTITY statorz;
ARCHITECTURE behav OF statorz IS
  QUANTITY z : REAL := 0.0;
  BEGIN
  z == sqrt((rs**2.0 + ((MATH_PI/30.0)*(p/2.0)* n_val*ls)**2.0));
  i == v/z;
END ARCHITECTURE behav;
```

The dc voltage drop in the three-phase rectifier circuit is accounted for by providing an equivalent diode with voltage drop of \(2V_d\), where \(V_d\) is the forward drop of a single diode in the actual rectifier.

The other models in the subsheet are as shown in the figure at the beginning of this section, and are obtained from the `basic vhdlams` library on the `AMS` tab. This circuit requires far less time for simulation than the detailed claw-pole alternator (HMIN=10µs, HMAX=1ms, TEND=60s).
Results

The graph outputs show the variation of the battery current, voltage and state of charge with respect to the velocity of the powertrain model. The battery voltage follows the speed characteristics of the powertrain model. The first Display Element shows the velocity output from the Powertrain model. The battery current and the rectified current are plotted in the second graph. The alternator outputs, are displayed in the third graph.
4.4 Multilevel Modeling Techniques: Loads

Concepts

This case study uses a load model to illustrate the concept of having multiple architectures with different behaviors defined within an individual model. It explains three different architectures that can be used to characterize a load used in the powernet system. In addition, the concept of component instantiation is described, using a lamp model to show how a particular load behavior can be reused in another model.

Background

The powernet system supports three different types of loads, each load being different only in its behavior and not its interface. The first type of load uses an admittance load behavior where the current in the model varies according to the control quantity in an ohmic manner. The second type of load uses a nominal load behavior, which is described with a nominal power and voltage specification. The third load behavior is for switched models where the control quantity switches a nominal load ON and OFF. This switched load behavior is then reused in a lamp model to illustrate component instantiation.

The following figure shows the basic circuit used for the load model with several architectures:

If the pins of two different data types are connected directly, a flexible OmniCaster will automatically be inserted between them. The flexible OmniCaster model will choose the correct transformation based on the data types of the pins connected to it.
Model: Load

Entity Description

The entity description indicates that three parameters are accepted by the load model: nominal power \( p_{\text{nom}} \) in Watts, nominal voltage \( v_{\text{nom}} \) in Volts, and ramp time \( t_{\text{ramp}} \) in seconds. It is not necessary to use all parameters specified in the **GENERIC** declaration within each architecture of the model.

The **ENTITY** description also defines two electrical ports, \( p \) and \( m \), for the load model. This declaration requires the use of the **IEEE.ELECTRICAL_SYSTEMS** package. A signal port for the control signal \( \text{on\_ctrl} \) of type REAL is also defined.

Architecture Description: Admittance Load

The **admittance** architecture models the current as proportionally dependent on the control signal value. The control signal value will be the admittance \( y \).

\[
i = y \cdot v
\]

This architecture uses the \( t_{\text{ramp}} \) parameter, but not \( v_{\text{nom}} \) or \( p_{\text{nom}} \).

This architecture first transforms the input \( \text{on\_ctrl} \) (REAL **SIGNAL** value) to \( \text{ctrl\_qty} \) (REAL **QUANTITY** value) using the \'RAMP\' attribute as follows:

\[
\text{ctrl\_qty} == \text{on\_ctrl}\text{^RAMP}(0.0,0.0);
\]

The \'RAMP\' attribute transforms a signal to a quantity which follows the corresponding value of the signal with the delay of the specified rise time and fall time. If these parameters are 0.0, then the value of the quantity follows that of the signal instantaneously. If the value of any parameter is greater than 0.0, the corresponding value change is linear from the current value of the signal to its new value, whenever that signal has an event.
The architecture then introduces a PT1 behavior on the control quantity to follow a first-order low-pass response and generates a new control quantity \textit{ctrl\_ramp}, according to the specified ramp time \( t\_ramp \). The following equation is used:

\[
\text{ctrl\_ramp}'\text{DOT} = (1.0/t\_ramp) \cdot (\text{ctrl\_qty} - \text{ctrl\_ramp});
\]

If the \textit{ctrl\_ramp} quantity is positive, it is used as the admittance value for the load model according to the following equation:

\[
\begin{align*}
\text{IF} & \quad \text{on\_ctrl} <= 0.0 \quad \text{USE} \\
& \quad i == 0.0; \\
\text{ELSE} & \quad i == v \cdot \text{ctrl\_ramp}; \\
\text{END USE};
\end{align*}
\]

Since the 'RAMP' attribute forces a synchronization between the analog and digital simulators, similar to the \textit{BREAK} statement, a separate \textit{BREAK} statement is not necessary for the \textit{on\_ctrl} signal.

The complete architecture description is as follows:

\[
\begin{align*}
\text{ARCHITECTURE} & \quad \text{admittance} \quad \text{OF} \quad \text{load} \quad \text{IS} \\
& \quad \text{QUANTITY} \quad v \quad \text{ACROSS} \quad p \quad \text{TO} \quad m; \\
& \quad \text{QUANTITY} \quad \text{ctrl\_ramp}; \quad \text{REAL} \quad := \quad 0.0; \\
& \quad \text{QUANTITY} \quad \text{ctrl\_qty}; \quad \text{REAL} \quad := \quad 0.0; \\
\text{BEGIN} & \\
& \quad \text{ctrl\_qty} \quad := \quad \text{on\_ctrl}'\text{RAMP}(0.0,0.0); \\
& \quad \text{ctrl\_ramp}'\text{DOT} \quad := \quad (1.0/t\_ramp) \cdot (\text{ctrl\_qty} - \text{ctrl\_ramp}); \\
& \quad \text{IF} \quad \text{on\_ctrl} \quad := \quad 0.0 \quad \text{USE} \\
& \quad \quad \quad i \quad := \quad 0.0; \\
& \quad \text{ELSE} \\
& \quad \quad \quad i \quad := \quad v \cdot \text{ctrl\_ramp}; \\
& \quad \text{END USE}; \\
\text{END ARCHITECTURE} \quad \text{admittance};
\end{align*}
\]

The \textit{ctrl\_ramp} and \textit{ctrl\_qty} are declared as free quantities within the admittance architecture. The input control admittance is provided by a triangular source time function. An OmniCaster model transforms the \texttt{REAL QUANTITY} output of the time function to a \texttt{REAL SIGNAL} value input for the load model. See also "Using Transformation Models" on page 18.
**Results**

The first Display Element shows the PT1 behavior of the transformed control signal. The second and third Display Elements show the variation of the voltage and current of the admittance load. The current peaks at 12 amperes when the admittance is 1 siemens. But there is a small voltage drop at peak load. The voltage does not recover completely when the load is removed, because the battery has discharged slightly.
Architecture Description: Nominal Load

The nominal architecture of the load entity models an ohmic load with a nominal power and voltage rating, the current being scaled by the control signal. This architecture uses all three declared parameters, nominal voltage ($v_{nom}$), nominal power ($p_{nom}$) and ramp time ($t_{ramp}$).

The Step source block provides the control signal input and an OmniCaster model is used to provide a REAL signal input to the load model. The $p_{nom}$ value used in this example is 50W, and the $v_{nom}$ value is 12V, corresponding to the system bus voltage. The nominal resistance is then 2.88Ω.

Similar to the admittance architecture, this architecture first transforms the input $on_{ctrl}$ (REAL SIGNAL) to $ctrl_{qty}$ (REAL QUANTITY) using the 'RAMP attribute, as follows:

$$ctrl_{qty} = on_{ctrl}^{'}RAMP(0.0,0.0);$$

The architecture then introduces a PT1 behavior on the control quantity to follow a first-order low-pass response and generates a new control quantity $ctrl_{ramp}$ according to the specified ramp time. The following equation is used:

$$ctrl_{ramp}^{'}OFT = (1.0/t_{ramp}) \times (ctrl_{qty} - ctrl_{ramp});$$

The nominal architecture differs from the admittance architecture in that for a positive value of $on_{ctrl}$, the current is modeled in terms of nominal resistance, as follows:

$$\text{IF } (on_{ctrl} <= 0.0) \text{ USE}$$
$$\quad i = 0.0;$$
$$\text{ELSE}$$
$$\quad i = (v/v_{nom}) \times ctrl_{ramp}^{'p_{nom}/v_{nom}};$$
$$\text{END USE;}$$

Since the 'RAMP attribute forces a synchronization between the analog and digital simulators similar to the BREAK statement, a separate BREAK statement is not necessary for the $on_{ctrl}$ signal.
The complete architecture description is as follows:

```vhdl-ams
ARCHITECTURE nominal OF load IS
  QUANTITY v ACROSS i THROUGH p TO m;
  QUANTITY ctrl_ramp: REAL := 0.0;
  QUANTITY ctrl_qty: REAL := 0.0;
BEGIN
  ctrl_qty <= on_ctrl'RAMP(0.0,0.0);
  ctrl_ramp'DOT <= (1.0/t_ramp) * (ctrl_qty - ctrl_ramp);
  IF (on_ctrl <= 0.0) USE
      i <= 0.0;
  ELSE
      i <= (v/v_nom) * ctrl_ramp* (p_nom/v_nom);
  END USE;
END ARCHITECTURE nominal;
```

The `ctrl_ramp` and `ctrl_qty` are declared as free quantities within the *nominal* architecture.

To change the architecture used on the sheet, double-click the *Load* symbol to open the Properties dialog, and click the «Library» tab. Choose the «nominal» architecture from the list «Select Architecture/Modeling Level». Click «OK» to apply the changes.

When a VHDL-AMS model from a library is first placed on the sheet, the default architecture is selected.
Results

The first Display Element illustrates the PT1 behavior of the transformed control signal. The second and third Display Elements show the variation of the current and voltage of the power load. Because of the voltage drop, actual power is less than the nominal 50W.

Architecture Description: Switched Load

The switch architecture models the on-off switch control for an ohmic load with a nominal power and voltage rating, the load current being further scaled by the control signal value. The Pulse time function provides the control signal input, and an OmniCaster model is used to provide a REAL signal input to the load model. The $p_{nom}$ value is 20W, and the $v_{nom}$ value is 12V, corresponding to the system bus voltage. This represents a constant resistance $r_{nom}$ of 7.2Ω.
The `switch` architecture does not transform the `on_ctrl` signal input to `ctrl_qty` quantity output using the `RAMP` attribute as the previous two architectures do. Instead, it uses an `IF-USE` statement to specify an ON/OFF condition of 0/1, as shown in the following code snippet:

```vhdl
IF on_ctrl <= 0.5 USE
  ctrl_qty := 0.0;
ELSE
  ctrl_qty := 1.0;
END USE;
BREAK ON on_ctrl;
```

To synchronize between the analog and digital statements, a separate `BREAK` statement must be introduced.

The architecture then introduces a PT1 behavior on the control quantity to follow a first order low pass response and generates a new control quantity called `ctrl_ramp` according to the specified ramp time. The following equation is used:

```vhdl
ctrl_ramp'DOT == (1.0/t_ramp) * (ctrl_qty - ctrl_ramp);
```

The `switch` architecture is similar to the `nominal` architecture in that for a positive value of `on_ctrl`, the current is modeled in terms of nominal resistance, as follows:

```vhdl
IF (on_ctrl <= 0.0) USE
  i := 0.0;
ELSE
  i := (v/v_nom) * ctrl_ramp * (p_nom/v_nom);
END USE;
```

The complete architecture description is as follows:

```vhdl
ARCHITECTURE switch OF load IS
  QUANTITY v ACROSS i THROUGH p TO m;
  QUANTITY ctrl_ramp: REAL := 0.0;
  QUANTITY ctrl_qty: REAL := 0.0;
BEGIN
  IF on_ctrl <= 0.5 USE
    ctrl_qty := 0.0;
  ELSE
    ctrl_qty := 1.0;
  END USE;
  ctrl_ramp'DOT == (1.0/t_ramp) * (ctrl_qty - ctrl_ramp);
  IF ctrl_ramp <= 0.0 USE
    i := 0.0;
  ELSE
    i := (v/v_nom) * ctrl_ramp * (p_nom/v_nom);
  END USE;
  BREAK ON on_ctrl;
END ARCHITECTURE switch;
```

The `ctrl_ramp` and `ctrl_qty` are declared as free quantities within the `switch` architecture.

To change the architecture used by a model, double-click the `Load` symbol to open the Properties dialog, and click the «Library» tab. Choose the «switch» architecture from the list «Select Architecture/Modeling Level». Click <OK> to apply the changes.
Case Studies

Results

The first Display Element illustrates the PT1 behavior of the transformed control signal. The second and third Display Elements show the variation in the voltage and current of the switched load. Because of the voltage drop, actual power is less than the nominal 20W.

Model: Lamp

This example illustrates the reuse of existing models within new models using component instantiation. The following example illustrates a Lamp model whose definition includes the switch architecture of the load model.
Entity Description

The interface of the lamp model is similar to that of the load model in the following ways: it accepts nominal voltage \((v_{nom})\), nominal power \((p_{nom})\), and ramp time \((t_{ramp})\) as parameter inputs, a control signal port input, and provides the electrical output between two pins, \(p\) and \(m\).

The voltage output from this model is also provided by the \(v_{out\ OUT}\) quantity port. This output value is intended to allow the animation of the lamp symbol.

The complete entity description is as follows:

```vhdl
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
ENTITY lamp IS
  GENERIC(
    p_nom: REAL := 20.0;
    v_nom: VOLTAGE := 12.0;
    t_ramp: REAL := 1.0e-5);
  PORT (
    p, m : ELECTRICAL;
    on_ctrl : IN REAL := 0.0;
    v_out : OUT VOLTAGE := 0.0);
END ENTITY lamp;
```

Architecture Description

The Lamp model instantiates only the model that is needed in its architecture description. This is in contrast to the component instantiation used in the Battery model (Method 2), where all models of the WORK library are available for use in an architecture. This methodology should be used if only a few models of the WORK library are needed. Every model that is visible in an architecture must be compiled before simulation, which lengthens the simulation time.

The load entity is instantiated with the component name \(lp\) and the switch architecture is associated with the component, using the following line:

```vhdl
lp: ENTITY WORK.load(switch)
```

This method of instantiation is called direct instantiation. There is no USE WORK.ALL declaration in the entity description. The generic values and the port values of the lamp model are mapped to those in the load model using the following lines in the architecture:

```vhdl
GENERIC MAP (p_nom => p_nom, v_nom => v_nom, t_ramp => t_ramp)
PORT MAP (p => p, m => m, on_ctrl => on_ctrl);
```

The complete architecture description of the lamp model is as follows:

```vhdl
ARCHITECTURE behav OF lamp IS
  QUANTITY \(v\) ACROSS p TO m;
BEGIN
  lp: ENTITY WORK.load(switch)
    GENERIC MAP (p_nom => p_nom, v_nom => v_nom, t_ramp => t_ramp)
    PORT MAP (p => p, m => m, on_ctrl => on_ctrl);
  v_out == v;
END ARCHITECTURE behav;
```
Results

The Display Elements show the voltage response of the lamp model when it is switched ON and OFF using the control signal. The first graph shows the control signal and the second graph the lamp voltage.
Symbol Animation

An Animated Symbol changes representation (shape, color) during the simulation process, depending on conditions applied to simulation quantities (for example, voltage, current, speed). The lamp circuit uses an Animated Symbol to illustrate the lamp states ON and OFF. If the control signal `on_ctrl ≤ 0.5`, the level 0 of the lamp symbol is displayed. If the control signal `on_ctrl > 0.5` and the output voltage `v_out > 0`, the levels 0 and 1 are displayed. In the case where `on_ctrl > 0.5` but `v_out ≤ 0`, the symbol displays its default level 0, indicating the lamp is OFF.

To obtain different symbol views, graphical elements must be placed on different symbol levels. The conditions defined in animations can only select between different levels in the symbol, not between different drawing elements.

The following steps describe how to implement simulation-dependent animation for the Lamp model symbol:

1. To create animated symbols for the model, right-click on Lamp (or my_lamp) in the model tree, and select «Edit Symbol». The Symbol Editor opens. Increase the magnification to 300%. Delete the text and the rectangle, select the filled circle icon from the toolbar, draw a circle and center it between the two terminals. Double-click the circle and change the fill color to a beige or another neutral color. Use the Text Element symbol on the toolbar to create the ‘+’ and ‘−’ symbols. The graphic just created is associated with Level 0 of the model. Copy the circle.

2. Make sure the Object Browser is displayed in the left pane and click the Add button in the Object Browser to create Level 1. Click the eye next to Level 0 to turn off display of the graphics and click the eye next to Level 1 to turn on its graphics. Click on the Level 1 folder and paste the circle onto the edit sheet. Change the circle’s color to red and position it in the same place as the circle in Level 0. Select FILE > SAVE.

3. To work with the symbol animation using an existing model, create an editable copy of the model by clicking on the «Users» tab of the ModelAgent, selecting the User library, and right-clicking on Packages. Choose INSERT MODEL(S) FROM VHDLA FILE... . Browse to the VHDL-AMS Tutorial folder. From the Tutorial Examples folder, select lamp.vhd. In the Insert Model window, check the box for my_lamp and click <OK>. The model will be added to the user library and will be editable.

4. Choose DRAW > ANIMATION in the Symbol Editor.

   1. Click the Add symbol twice to create two new expressions for the symbol animation.

   2. Define the condition `on_ctrl ≤ 0.5` in the first «Expression» field, define the condition `on_ctrl > 0.5 and v_out > 0.0` in the second.
The model parameters \texttt{on\_ctrl} and \texttt{v\_out} control the symbol view. Make certain the parameter names are used correctly. Click on the button and open an additional input dialog where all parameters of a model are listed.

5 Level 0 is automatically selected for all conditions. Click on the \texttt{<Activate Level>} button for the second condition, and select Level 1. Click \texttt{<OK>} to apply the changes.

6 Choose \texttt{FILE > SAVE}, then exit the Symbol Editor.
4.5 Multidomain System Modeling: Linear Drive System, Solenoid

Concepts

This case study uses an electromechanical subsystem in a vehicle powernet to illustrate multidomain modeling. The concept of subsheet modeling is also introduced in this case study. The first example in this case study illustrates the modeling of linear drive systems that might be used, for example, in a power window system. The second example illustrates the modeling of a simple solenoid that may be used to control hydraulic valves in a vehicle.

4.5.1 Linear Drive System

Background

This linear drive system is powered by the SoC Battery model and is connected to the battery through a switch. The switch control is provided through a pulse wave time function that is passed through a G(s) block to model the PT1 (first-order low-pass response) switching behavior. The motor model is described in a VHDL-AMS subsheet that can be created by either typing out the entity and architecture of the model in a new text subsheet, or by copying the model description from the `motor.vhd` file. A free-wheeling diode model is connected in parallel to the motor model to limit the motor voltage, so that the switch is protected when the battery is switched off.

Note that the student version of SIMPLORER cannot use some of the elements that are used to provide the motor voltage through the switch. After opening this case study, select the `bat_soc1`, `c1`, `wire1`, `ctrl`, `pt1`, and `switch` elements, right-click and select «Don’t Add to Model Description» to deactivate these elements. To use the hatched element `e1` in place of the other elements, right-click and select «Don’t Add to Model Description» to clear the checkbox and activate it.

The rotational mechanical output of the motor model drives a rotational mass that models the inertia. The output from the mass then passes through a gear box model. This gearbox element can be copied from the `gearbox.vhd` file or typed out in a new subsheet element. The translational output of the gear box model displaces the translational mass, the amount of displacement is controlled by a limit stop model. A spring and damper combination is included in the translational domain.
Entity Description

The motor model has three conservative pins: \( n1 \) and \( n2 \) are electrical inputs and \( ROT \) is the mechanical output. The model also accepts the armature resistance as an input quantity \( ra \), and provides the rotor angle \( phi \) and rotor speed \( n \) as outputs. Because the model defines \( ra \) as an input \texttt{QUANTITY}, it could vary during simulation. None of the other inputs (\( ls, ke, j, ia0, n0, phi0 \)) can vary during simulation. The other model parameters used in the model are listed in the following table:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Name</th>
<th>Property</th>
<th>Default Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC</td>
<td>la</td>
<td>INDUCTANCE</td>
<td>0.01</td>
<td>100m</td>
<td>Armature Inductance [H]</td>
</tr>
<tr>
<td></td>
<td>ke</td>
<td>FLUX</td>
<td>1.0</td>
<td>0.5</td>
<td>Rotor Flux Constant [V]</td>
</tr>
<tr>
<td></td>
<td>j</td>
<td>MOMENT_INERTIA</td>
<td>0.075</td>
<td>0.075</td>
<td>Rotor Moment of Inertia [kg*m²]</td>
</tr>
<tr>
<td></td>
<td>ia0</td>
<td>CURRENT</td>
<td>0.0</td>
<td>0.0</td>
<td>Initial Armature Current [A]</td>
</tr>
<tr>
<td></td>
<td>n0</td>
<td>VELOCITY</td>
<td>0.0</td>
<td>0.0</td>
<td>Initial Armature Speed [rpm]</td>
</tr>
<tr>
<td></td>
<td>phi0</td>
<td>ANGLE</td>
<td>0.0</td>
<td>0.0</td>
<td>Initial Rotor Position [rad]</td>
</tr>
<tr>
<td>QUANTITY</td>
<td>ra</td>
<td>IN_REAL</td>
<td>1.0</td>
<td>1.2</td>
<td>Armature Resistance [Ω]</td>
</tr>
<tr>
<td></td>
<td>n, phi</td>
<td>OUT_REAL</td>
<td>0.0</td>
<td>0.0</td>
<td>Speed [rpm], Rotor Angle [rad]</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>n1, n2</td>
<td>ELECTRICAL</td>
<td></td>
<td></td>
<td>Electrical Terminals</td>
</tr>
<tr>
<td></td>
<td>rot</td>
<td>ROTATIONAL_V</td>
<td></td>
<td></td>
<td>Mechanical Terminal</td>
</tr>
</tbody>
</table>

The equivalent VHDL-AMS description for defining the model interface is as follows:

```vhdl-ams
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MECHANICAL_SYSTEMS.ALL;
USE IEEE.MATH_REAL.MATH_PI;
ENTITY motor IS
  GENERIC IS
    la : INDUCTANCE := 0.01;
    ke : FLUX := 1.0;
    j : MOMENT_INERTIA := 0.075;
    ia0 : CURRENT := 0.0;
    n0 : VELOCITY := 0.0;
    phi0 : ANGLE := 0.0;
  END GENERIC;
  PORT TERMINAL n1, n2 : ELECTRICAL;
  TERMINAL rot : ROTATIONAL_V;
  QUANTITY ra : IN_RESISTANCE := 1.2;
  QUANTITY n : OUT VELOCITY;
  QUANTITY phi : OUT ANGLE;
END ENTITY motor;
```

The model needs to use the electrical_systems and mechanical_systems packages of the IEEE library in order to use the ELECTRICAL and MECHANICAL conservative pins, respectively. Additionally, since the model architecture requires \( \pi \), the math_real package needs to be used.
Architecture Description

For conversions between translational and rotational values in the mechanical domain, the model architecture defines the following two constants:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>n2om</td>
<td>$2.0 \cdot \pi / 60.0$</td>
</tr>
<tr>
<td>om2n</td>
<td>$60.0 / (2.0 \cdot \pi)$</td>
</tr>
</tbody>
</table>

The across and through quantities for the conservative pins are defined as follows:

<table>
<thead>
<tr>
<th>Conservative Pins</th>
<th>Across</th>
<th>Through</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1, n2</td>
<td>v</td>
<td>i</td>
</tr>
<tr>
<td>rot</td>
<td>omega</td>
<td>torque</td>
</tr>
</tbody>
</table>

The DC motor behavior is based on the following equations:

\[
\phi = L_a \cdot i \\
\frac{\partial (\text{angle})}{\partial t} = \omega \\
\frac{\partial \omega}{\partial t} = \frac{(\text{mi} + \text{incertia})}{j} \\
v = \frac{i \cdot r_a + \phi}{r_a} + k_c \cdot \omega
\]

The speed and position of the rotor are output through n and phi. Three free quantities angle, flux, and mi are declared within the model architecture. Initial values are provided to angle, omega, and flux quantities using the BREAK statement.

The equivalent VHDL-AMS description for defining the model architecture is as follows:

```vhdl-ams
ARCHITECTURE behav OF motor IS
  QUANTITY v ACROSS i THROUGH N1 TO N2;
  QUANTITY omega ACROSS torque THROUGH rot TO rotational_v_ref;
  QUANTITY local_mi, local_angle, local_flux : REAL;
  CONSTANT n2om : REAL := 2.0 * math_pi / 60.0;
  CONSTANT om2n : REAL := 60.0 / (2.0 * math_pi);
BEGIN
  BREAK
  local_angle => phi0,
  omega => n0 * n2om,
  local_flux => ia0 * la;
  v == i * ra + local_flux'DOT + ke * omega;
  local_mi == ke * i;
  omega'DOT == (1.0 / j) * (local_mi + torque);
  local_angle'DOT == omega;
  local_flux == i * la;
  n == omega * om2n;
  phi == local_angle;
END ARCHITECTURE behav;
```
Case Studies

Model: Gearbox

Entity Description
This model accepts rotational input (velocity-torque representation) and provides output on a translational (displacement-force representation) pin. The gear radius is defined as a GENERIC REAL input value. The parameters used in the model are listed in the following table:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Name</th>
<th>Property</th>
<th>Default Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC</td>
<td>radius</td>
<td>REAL</td>
<td>0.1</td>
<td>0.025</td>
<td>Gear radius [m]</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>rot</td>
<td>ROTATIONAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trans</td>
<td>TRANSLATIONAL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The equivalent VHDL-AMS description for defining the model interface is as follows:

```vhdl
LIBRARY IEEE;
USE IEEE.MECHANICAL_SYSTEMS.ALL;
ENTITY gearbox IS
  GENERIC(radius  : REAL := 0.1);
  PORT(TERMINAL rot : ROTATIONAL_V;
       TERMINAL trans : TRANSLATIONAL);
END ENTITY gearbox;
```

Architecture Description
The gearbox model transforms the rotational input (velocity-force) to translational output (distance-torque representation) based on the specified gear radius. The across and through quantities for the conservative pins are defined as follows:

<table>
<thead>
<tr>
<th>Conservative Pins</th>
<th>Across</th>
<th>Through</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROT, ROTATIONAL_V_REF</td>
<td>omega</td>
<td>torque</td>
</tr>
<tr>
<td>TRANS, TRANSLATIONAL_REF</td>
<td>position</td>
<td>force</td>
</tr>
</tbody>
</table>

The transformation is performed by the following two equations:

\[
\frac{\partial}{\partial t} \text{position} = \omega \cdot r \\
\tau = -\text{Force} \cdot r
\]

The equivalent VHDL-AMS description for the model architecture is as follows:

```vhdl
ARCHITECTURE behav OF gearbox IS
  QUANTITY omega ACROSS torque THROUGH rot TO ROTATIONAL_V_REF;
  QUANTITY position ACROSS force THROUGH trans TO TRANSLATIONAL_REF;
BEGIN
  position\'DOT = omega * radius;
  torque = -force * radius;
END ARCHITECTURE behav;
```

The translational force output displaces the translational mass of 1 kg (this might represent the power window in a vehicle). The displacement of the mass is limited to 50cm by the Limit Stop model. Appropriate damping/spring rate coefficients are set for Limit Stop, as well as for other mechanics in the TRANSLATIONAL domain.
Results

The first graph shows the PT1 behavior of the control signal modeled with a Laplace Transform. The second graph shows the motor current and position characteristics. The third graph shows the displacement of the mass, limited at an upper limit of 0.5m, and the damping effects. The fourth graph shows the battery voltage and state of charge characteristics. The ‘window’ closes at about 1.5 sec, but the motor is kept energized for another 3.5 sec.
4.5.2 Solenoid System

Background

SIMPLORER allows the use of finite element data generated from a Maxwell circuit simulation through the ECE (Equivalent Circuit Export) link. In this example, two solenoid models are simulated. The first model uses finite element data from the ECE link (Maxwell model) and the second uses an approximated model developed using VHDL-AMS (VHDL-AMS model).

The solenoid is powered by the SoC battery and is controlled by a switch that functions through a pulse model. Each solenoid model has two electrical pins for input, and two translational pins for output. Note that solenoid models can also convert energy in the other direction, mechanical to electrical. The mechanical force, measured using a force meter, is used to displace a translational mass representing the plunger. The opposing force of gravity on the plunger is represented by a force \((F = \text{mass} \times 9.8 \text{m/s}^2)\), and the displacement of the plunger is controlled by a limit stop model.

The VHDL-AMS model uses an idealized equation based on the input current value and the position of the plunger to determine the flux linkage in the solenoid and the corresponding force on the plunger. The Maxwell model is more accurate since it uses the finite element data that models fringing and saturation effects.
Model: VHDL-AMS Solenoid

Entity Description

The VHDL-AMS solenoid is described by equations derived in Woodson and Melcher: *Electromechanical Dynamics* [4]. The *Solenoid* model accepts three parameters: the maximum inductance value per turn at minimum air gap $L_0$, an inductance coefficient $K$, and the number of coil turns $N$. These parameters are used for the computation of the instantaneous inductance value, as follows ($x$ is the displacement of the plunger):

$$L(x) = \frac{N^2 \cdot L_0}{1 + Kx}$$

The gap $x$ begins at 0.0127m (0.5in) and ends at 0.0, when the solenoid is fully closed.

The value of $K$ might be derived from analyzing the magnetic circuit, from measurement, or in this case, from finite element solutions. But given that $L(x)$ follows this equation, the coil voltage is:

$$V = \frac{dx}{dt} = \frac{d}{dt}(L \cdot i) = N^2 \cdot L_0 \cdot \frac{d}{dt}\left(\frac{-i}{1 + Kx}\right)$$

Both $x$ and the coil current $i$ may vary with time. Carrying out the derivatives:

$$V = N^2 \cdot L_0\left\{\frac{i}{1 + Kx} \cdot \frac{dt}{dt} - \frac{iK}{1 + Kx^2} \cdot \frac{dx}{dt}\right\}$$

The $di/dt$ term is a transformer voltage, and the $dx/dt$ term is a speed voltage.

On the mechanical side, an energy conversion process generates a force according to:

$$f = \frac{d}{dx} W_m' = W_m' = \frac{1}{i} \int_0^i \lambda(i', x) \, di'$$

where $W_m'$ is the magnetic coenergy and $i'$ is a dummy variable for integration.

If the core material is linear, then the magnetic energy and coenergy are the same. Substituting for $L$ and then carrying out the integration:

$$\lambda(i', x) = \frac{N^2 \cdot L_0 \cdot i'}{1 + Kx} \quad W_m' = \frac{N^2 \cdot L_0 \cdot i^2}{1 + Kx \cdot \frac{1}{2}}$$

The force is calculated by taking a partial derivative with respect to $x$:

$$f = \frac{N^2 \cdot L_0 \cdot i^2}{2} \frac{\partial}{\partial x}\left\{\frac{1}{1 + Kx}\right\} \quad f = \frac{N^2 \cdot L_0 \cdot K \cdot i^2}{2 \cdot (1 + Kx)}$$

The negative sign means that $f$ acts to decrease $x$, for any current $i \neq 0$.

The VHDL-AMS model uses flux, so that the following simplifications are possible:

$$\lambda = \frac{N^2 \cdot L_0 \cdot i}{1 + Kx} \quad f = -\frac{\lambda^2 \cdot K}{2 \cdot L_0 \cdot N^2} \quad V = \frac{dx}{dt}$$
The force is $\lambda^2$ times a constant called $F_k$. The term $N^2 L_0$ appears in two equations, and it is a constant. Therefore:

$$L_{\text{max}} = N^2 \cdot L_0$$

$$\lambda = \frac{L_{\text{max}} \cdot i}{1 + Kx}$$

$$F_k = \frac{K}{2 \cdot L_{\text{max}}}$$

$L_{\text{max}}$ is the maximum inductance, which occurs when $x$ is zero.

The VHDL-AMS model includes two external and two mechanical pins. Given a current $i$ and position $x$, the model equations calculate $\lambda$ and $f$. The value of $f$ will be passed to the mechanical pins as a through variable. The quantity $d\lambda/dt$ appears across the electrical pins.

One might follow a similar process to develop the VHDL-AMS models for other multidomain components. First, the equations for stored energy in each of the domains, in terms of the through and across variable, must be obtained. Then, the energy (or coenergy) is equated. Lossy elements could be added separately.

The model accepts a current value from electrical pins $p$ and $m$, and a position value from the mechanical pins, $\text{pos1}$ and $\text{pos2}$. The parameters used in the model are listed in the following table:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Name</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC</td>
<td>$L_0$</td>
<td>INDUCTANCE</td>
<td>1.25 e-7</td>
<td>Max inductance per turn at min gap [H]</td>
</tr>
<tr>
<td></td>
<td>$K$</td>
<td>REAL</td>
<td>197.735</td>
<td>Inductance coefficient</td>
</tr>
<tr>
<td></td>
<td>$N$</td>
<td>REAL</td>
<td>1.0</td>
<td>Number of coil turns</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>$p, m$</td>
<td>ELECTRICAL</td>
<td></td>
<td>Electrical pins</td>
</tr>
<tr>
<td></td>
<td>$\text{pos1}$</td>
<td>TRANSLATIONAL</td>
<td></td>
<td>Mechanical pins of translational domain</td>
</tr>
<tr>
<td></td>
<td>$\text{pos2}$</td>
<td>TRANSLATIONAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The equivalent VHDL-AMS description for defining the model interface is as follows:

```
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
USE IEEE.MECHANICAL_SYSTEMS.ALL;
ENTITY solenoid IS
  GENERIC (L0 : INDUCTANCE := 1.25e-7;
            K : REAL := 197.735;
            N : REAL := 1.0);
  PORT (TERMINAL p, m : ELECTRICAL;
         TERMINAL pos1, pos2 : TRANSLATIONAL);
  QUANTITY force_out : OUT REAL := 0.0);
END ENTITY solenoid;
```
Architecture Description

The across and through quantities for the conservative pins are defined as follows:

<table>
<thead>
<tr>
<th>Conservative Pins</th>
<th>Across</th>
<th>Through</th>
</tr>
</thead>
<tbody>
<tr>
<td>p, m</td>
<td>voltage</td>
<td>current</td>
</tr>
<tr>
<td>pos1, pos2</td>
<td>position</td>
<td>force</td>
</tr>
</tbody>
</table>

The equations for this model are as follows:

\[
L_{\text{max}} = N^2 \cdot L_0 \\
L = \frac{L_{\text{max}}}{1 + Kx} \\
\lambda = L \cdot i \\
v = \frac{\partial \lambda}{\partial t} \\
F = -\lambda^2 \cdot F_K
\]

The equivalent VHDL-AMS description for defining the model interface is as follows:

```vhdl-ams
ARCHITECTURE behav OF solenoid IS
    CONSTANT Lmax : INDUCTANCE := L0 * N * N;
    CONSTANT Fk : FORCE := K / (2.0 * Lmax);
    QUANTITY v ACROSS i THROUGH p TO m;
    QUANTITY position ACROSS force THROUGH pos1 TO pos2;
    QUANTITY L : INDUCTANCE;
    QUANTITY flux : FLUX;
BEGIN
    IF (position > 0.0) USE
        L == Lmax / (1.0 + K * position);
    ELSE
        L == Lmax;
    END USE;
    flux == L * i;
    v == flux'DOT;
    force == flux * flux * Fk;
    force_out == -force;
END ARCHITECTURE behav;
```

In this example, N=12500 turns, but the constant K is not readily known. It may come from a magnetic circuit analysis of the device, or it may come from two measurements at different gap spacings. In this example finite element solutions at two values of x are used:

<table>
<thead>
<tr>
<th>x</th>
<th>i</th>
<th>L</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0</td>
<td>0.084</td>
<td>1.251e-7</td>
<td>-27</td>
</tr>
<tr>
<td>3.0127</td>
<td>0.084</td>
<td>3.56e-8</td>
<td>-0.84</td>
</tr>
</tbody>
</table>

Here the value of i comes from an excitation of 1050 ampere-turns, and L is the inductance per turn.

For one approach, we could choose K to match L at both values for x. This should represent the electrical side better, over the whole range of plunger travel.

\[
K = \frac{1.251e-7}{1 + 0.0127 \cdot K} = 197.735
\]
Another approach is to match both $L$ and $I$, when $x$ is zero.

\[
\begin{align*}
L_{\text{max}} &= 12500 \cdot 1.251 \times 10^{-7} = 19.547 \\
\lambda &= L_{\text{max}} \cdot i = 19.547 \cdot 0.084 = 1.642 \\
F_K &= \frac{27}{1.642^2} = 10 \\
K &= 2 \cdot F_K \cdot L_{\text{max}} = 2 \cdot 10 \cdot 19.547 = 391.52
\end{align*}
\]

These two values of $K$ are quite different. It is not possible to match both $L$ and $I$ over the whole operating range because of the solenoid’s departure from ideal behavior.

In this example, we choose the lower $K$ value, because we want to represent the coil current as accurately as possible. If the mechanical side is of more concern, choose the higher $K$ value or a compromise value (300).

The VHDL-AMS solenoid model is limited since it assumes that the core material is linear, and ignores the effects of flux fringing. A better model might result from finite element analysis.
Results

The characteristics plotted in the Display Elements show the behavior of the circuit using the VHDL-AMS solenoid when the battery is connected to the circuit for a period of 1 sec.

The first graph shows coil current and position of the mass model. The second graph shows the input switch signal and the force on the plunger.

The mass model is moved from an initial position of 0.0127m to a final position of 0m within 0.1 sec, thereby closing the plunger gap. When the battery is disconnected from the circuit after 1 sec, the force of gravity causes the plunger to open again.

When the solenoid is first energized, the coil current increases with a fast L/R time constant because L is minimum at the open position. When the plunger begins to close, it generates a back EMF that causes a transient dip in coil current. When the solenoid is closed and L is maximum, the current increases with a slow time constant. The final coil current is determined by the battery voltage and coil resistance.

When the switch opens at 1 sec, free-wheeling diodes are necessary to permit gradual decay of the inductive coil current. When the current has been delayed enough, gravity overcomes the electromagnetic force, and the solenoid begins to open. As the plunger moves, it generates a back EMF that causes a transient increase in the coil current.
4.6 Mixed-Signal Modeling: DC-DC Model with PWM, Automotive Alarm System

Concepts

This case study illustrates several facets of digital modeling. A VHDL-AMS model of a Pulse-Width Modulator (PWM) controlling a simple DC-DC boost converter circuit illustrates digital circuit modeling. An automotive alarm system illustrates the use of a stimulus generator, and shows how to export VHDL-AMS models and use foreign models.

4.6.1 DC-DC Model

Background

The boost converter in the example steps up a 5V ideal source to a 42V battery-fed system. A PWM control based on the actual and reference current values switches an ideal BJT transistor. The focus of this example is the development of the PWM controller in VHDL-AMS.

A reference clock signal (cond1) is created within the PWM controller block with a period of 20µs and a duty cycle of 0.0105. A second signal (cond2) is created within the PWM model every time the current in the inductor l1 exceeds a reference value of 100A. The PWM controller block outputs an ON control signal if both the reference clock is ON (cond1) and the actual inductor current is less than the reference value (NOT cond2). Consequently, the control signal for the transistor is pulse-width modulated. It is switched on for the period of time when the condition (cond1 AND NOT cond2) is true.
Model: PWM Controller

Entity Description

The PWM controller modeled for this example accepts three parameters: a reference clock period (period), a duty cycle value for the reference clock (dc), and a maximum reference current value for the inductor current (ind_i_max). It has two port quantities: the actual inductor current value (ind_i) and a control signal output for switching the BJT (ctrl). Default values are provided for all generic and port values.

The equivalent VHDL-AMS description for defining the model interface is as follows:

```vhdl-ams
ENTITY pwm IS
  GENERIC(
    period : REAL := 1.0e-3; -- clock period
    dc : REAL := 0.5; -- duty cycle
    ind_i_max : CURRENT := 100.0); -- reference inductor current
  PORT(
    QUANTITY ind_i : IN CURRENT := 0.0; -- Inductor Current
    QUANTITY ctrl : OUT REAL := 0.0); -- Control Signal Output
END ENTITY pwm;
```

Architecture Description

The model architecture is divided into two basic functional parts that are represented as processes. Constant values are declared for time_period, hi_period, and lo_period of the reference clock signal, based on the specified period and duty cycle of the signal, as follows:

\[
\text{hi}_{\text{period}} = \text{dutycycle} \cdot \text{timeperiod} = \text{dc} \cdot \text{period} \\
\text{lo}_{\text{period}} = \text{timeperiod} - \text{hi}_{\text{period}}
\]

Two BOOLEAN signals cond1 and cond2 are used along with a REAL signal on_ctrl. The second signal cond2 uses the 'ABOVE' attribute to identify if the actual current value ind_i has exceeded the maximum reference current value ind_i_max.

```vhdl-ams
cond2 <= ind_i'ABOVE(ind_i_max);
```

The previous statement is equivalent to the pseudocode

\[\text{IF} (\text{actual current} > \text{maximum reference current}) \text{ THEN } \text{cond2} <= \text{TRUE}.\]

A process statement specifies the sequential behavior of a model. It can be triggered by a set of signals, conditions, or at specific time instances. The first process in the model (ref_clock) outputs the reference clock signal as a BOOLEAN value (cond1: TRUE=>ON) according to the specified values of lo_period and hi_period. It uses a LOOP statement along with a WAIT statement to model the output BOOLEAN clock signal. The WAIT statement suspends the process for a specific period of time. The following figure shows this PWM clock signal.

```vhdl-ams
ref clock : PROCESS
BEGIN
  LOOP
    cond1 <= FALSE;
    WAIT FOR lo_period;
    cond1 <= TRUE;
    WAIT FOR hi_period;
  END LOOP;
END PROCESS ref_clock;
```
The second process, called \texttt{ctrl}, has a sensitivity list of the signals that affect the process. In this case, the process is sensitive to events that occur on \texttt{cond1} and \texttt{cond2} signals. This means that the sequential \textbf{IF} statement within the process is executed whenever either \texttt{cond1} or \texttt{cond2} changes value from \texttt{TRUE} to \texttt{FALSE} or vice versa. The \textbf{IF} statement in the model implements the following pseudocode:

\begin{verbatim}
IF (actual current > maximum reference current) THEN control <= OFF (Switch BJT OFF)
ELSE
  IF (actual current < maximum reference current) AND (reference clock is ON)
    THEN control <= ON (Switch BJT ON)
  END IF;
END PROCESS pwm_ctrl;
\end{verbatim}

If neither condition holds true, then \texttt{ctrl_sig} remains unchanged. The BJT can be switched ON at each clock pulse shown in the previous figure, unless the current already exceeds the reference level. The BJT will then switch OFF when the current reaches the reference level.

The digital control signal is transformed to an analog quantity using the \texttt{RAMP} attribute. The equivalent VHDL-AMS description for defining the model interface is as follows:

\begin{verbatim}
ARCHITECTURE behav OF pwm IS
  CONSTANT time_period : TIME := 1 sec*period;
  CONSTANT hi_period : TIME := 1 sec*(dc*period);
  CONSTANT lo_period : TIME := time_period - hi_period;
  SIGNAL cond1, cond2 : BOOLEAN := FALSE;
  SIGNAL ctrl_sig : REAL := 0.0;
BEGIN
  cond2 <= ind_i'ABOVE(ind_i_max);
  ref_clock : PROCESS
  BEGIN
    LOOP
      cond1 <= FALSE;
      WAIT FOR lo_period;
      cond1 <= TRUE;
      WAIT FOR hi_period;
    END LOOP;
  END PROCESS ref_clock;
  pwm_ctrl : PROCESS (cond1,cond2)
  BEGIN
    IF (cond2) THEN
      ctrl_sig <= 0.0;
    ELSIF (cond1 AND (NOT cond2)) THEN
      ctrl_sig <= 1.0;
    END IF;
  END PROCESS pwm_ctrl;
  ctrl <= ctrl_sig'RAMP(0.0,0.0,0);
END ARCHITECTURE behav;
\end{verbatim}
Results

The first graph shows the control output of the PWM controller and the corresponding voltage of the load resistor. The outputs are pulse-width modulated. The second graph shows the variation of the actual inductor current which is being controlled with respect to the maximum allowable inductor current. The third graph shows the variations of the battery state of charge as a consequence of the switching action of the PWM controller.

The BJT is switched ON at each clock pulse during the first 3ms, as the current builds to 100A. Then the BJT begins to switch OFF at a variable time during each clock period. The BOOST converter begins to charge the battery each time the BJT switches OFF.
4.6.2 Automotive Alarm System

Background

The automotive alarm system is a simplified example of an electronic vehicle security protection system. It illustrates modeling concepts for multidomain sensors and digital logic and the use of a stimulus generator to create stimulus input patterns.

The system has three main parts:

- Multidomain microswitch sensor models that detect alarm conditions and provide control signals indicating that an alarm situation has occurred
- Digital stimulus generator that provides the remote key switch and digital vector sensor inputs
- Digital controller that arms the vehicle and provides a siren output if the security of an armed vehicle has been compromised

This system uses two types of mechanical microswitches (simple and advanced) that act as linear position sensors. The mechanical inputs for the sensor models are connected to translational domain position source models. These mechanical domain source models use a combination of step inputs to generate a pulse characteristic for the displacement value. The sensor model outputs vary as analog quantities and are converted to digital signals by OmniCaster models. Note that OmniCasters are automatically inserted when a connection is made between the analog control output of the sensor and the digital control input of the digital controller.

In order to simulate the example as-is, a license for the SIMPLORER Sensor Library is required. If this license is not available, the example can be simulated by deactivating the advanced_microswitch model on the sheet and enabling the simple_microswitch2 model. This can be done by choosing the «Don’t Add to Model Description» menu item from the model’s shortcut menu.
Model: Simple Microswitch

The first microswitch model (simple_microswitch) is a simplified sensor and is developed as a text subsheet in VHDL-AMS. It accepts a threshold position parameter and a varying position input. It determines when the sensor position has crossed the threshold and outputs a control quantity that is used by the digital controller.

Entity Description

This is a mechanical model and consequently, includes the mechanical_systems package. It accepts the position_threshold parameter as a generic input, and the mechanical input through a conservative TRANSLATIONAL domain pin. When the mechanical displacement on this conservative pin exceeds the parameter value of position_threshold, the control_output quantity signal is activated.

```
LIBRARY IEEE;
USE IEEE.MECHANICAL_SYSTEMS.ALL;
ENTITY simple_microswitch IS
  GENERIC(
    position_threshold : DISPLACEMENT := 0.5e-3);
  PORT(
    TERMINAL mech_input : TRANSLATIONAL;
    QUANTITY control_output : OUT REAL);
END ENTITY simple_microswitch;
```

Architecture Description

The model architecture uses the ‘ABOVE attribute to determine the threshold crossing and an IF-USE statement to set the control quantity output.

```
ARCHITECTURE behav OF simple_microswitch IS
  QUANTITY s_val ACROSS mech_input TO TRANSLATIONAL_REF;
  SIGNAL crossing : BOOLEAN := FALSE;
BEGIN
  crossing <= s_val'ABOVE(position_threshold);
  BREAK ON crossing;
  IF (crossing) USE
    control_output == 1.0;
  ELSE
    control_output == 0.0;
  END USE;
END ARCHITECTURE behav;
```

Model: Advanced Microswitch

The second sensor (advanced_microswitch) is a slightly more complicated model developed as a graphical subsheet with a microswitch component from the SIMPLORER Sensor Add-on library.

The Sensor library offers a wide variety of sensor models with a choice of using system-level models with idealized behavior, as well as device-level models with non-ideal behavior such as temperature sensitivity, nonlinearity, hysteresis, and measurement error. In general, all the models from the sensor library offer "ready parameterization," i.e., they can be easily parameterized by using datasheet information. Additionally, the library includes a complete set of building blocks that enable users to create custom sensor models very quickly using an easy-to-use, color-coded format.
The advanced microswitch is modeled as a graphical subsheet and uses a Level 2 Sensor Model from the SIMPLORER Sensors AddOns Library on the «AddOns» tab of the ModelAgent. The model is available from Level 2 Sensor Models>Linear Position>Displacement-Force>Microswitch, and the graphical subsheet can be modeled as shown below. Note that except for the microswitch model, all other elements are from the basic_vhdلامns library in the «AMS» tab of the ModelAgent.

This component accepts position input from a conservative mechanical terminal and toggles from its normally closed terminal to its normally open terminal when the specified threshold position is exceeded by some element in the displacement-force translational domain. The voltage output from this model’s conservative electrical terminals provides a control quantity that is used by the digital controller. The advanced_microswitch model is a more detailed model when compared to the simple_microswitch model since it also models switch resistances, and an additional response time.

In this example, the result of an additional response time within the sensor can be simulated using the advanced_microswitch model. This is explained further in “Results” on page 125.

Model: Stimulus Generator

The stimulus generator is used to provide one or more digital stimuli of types BIT, STD_LOGIC, BIT_VECTOR and STD_LOGIC_VECTOR. It provides an easy user interface for specifying the characteristics of digital stimuli and automatically generates VHDL source code for the user-specified stimuli. This model is available from the Digital Sources folder on the «Digital» tab of the ModelAgent.

The first time the Properties dialog of the stimulus generator model is opened, a time initialization dialog is displayed. The stimulus generator uses a base clock signal as a reference for creating events for other signals. Users can specify the time resolution, stimulus end time, and base clock characteristics in this dialog. After exiting the time initialization dialog, users can set up one or more individual digital stimuli that are required in the Schematic.

The events on each stimulus can be specified by setting the value of the signal on a time line or by specifying a pattern for the stimulus.
In this example, the stimulus generator is used to provide the key switch input as well as vector input from four sensors to the digital controller of the alarm system. To parameterize the stimulus generator, choose Properties from the model's shortcut menu. The time initialization dialog is parameterized with the following values:

Click the Add button to add two additional parameters. Add a parameter of type BIT and change the name to key. Add a bit vector parameter and keep the default name bvrsig1.

The key switch input is modeled as a BIT signal with signal events defined as follows:

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-30</td>
<td>0</td>
</tr>
<tr>
<td>30-250</td>
<td>1</td>
</tr>
<tr>
<td>250-340</td>
<td>0</td>
</tr>
<tr>
<td>340-550</td>
<td>1</td>
</tr>
<tr>
<td>550-620</td>
<td>0</td>
</tr>
<tr>
<td>620-850</td>
<td>1</td>
</tr>
<tr>
<td>850-910</td>
<td>0</td>
</tr>
<tr>
<td>910-1000</td>
<td>1</td>
</tr>
</tbody>
</table>
Set the values from the drop-down menu for each 10 ms time interval of the key parameter according to the table above.

The sensor bus input is modeled as a BIT_VECTOR with signal events on the vector defined by an alternate pattern.

Click on the symbol to open the pattern wizard. An alternate pattern is used to toggle two sets of values on the bit vector signal. The pattern repetition number specifies the number of times the pattern needs to be repeated. The size of the bit vector is specified in the Vector Length edit box and actual patterns are specified in the Value 1 and Value 2 section. The period for which each pattern should be applied on the bit_vector signal is specified in terms of the number of base clock pulses, in the Pulse Width Value 1 and Value 2 sections.

The the key switch and the sensor bus pins from the stimulus generator must be explicitly displayed using either the shortcut menu option, or by checking the pin display option in the Properties dialog so that they can be connected to the digital controller model.

Model: Digital Controller

The digital controller accepts a key switch input used to arm and disarm the vehicle, as well as inputs from the different sensors monitoring the security of the vehicle. Two position sensor inputs are available from the two microswitch models and a sensor bus input is available from the stimulus generator model. If any of the sensors detect a security infringement when the system is armed, the digital controller sends a siren output signal.

The logic for the digital controller is modeled as follows. The system is armed by providing an enable (a digital 1 bit) in the key switch input. The controller requires 30 ms to arm the system. Once the system is armed, triggering any of the sensors would cause the siren to turn on after another 30 ms. If the siren has been turned on then it can be switched off instantaneously by disabling the key switch (a digital 0 bit). Note that it is possible to arm a system even if sensors are already triggered (for example, if a door has not been closed properly). In this case, the siren will be turned on 30 ms after the key switch is enabled.
Entity Description

The digital controller is modeled as a VHDL-AMS text subsheet and the entity declaration for
the model is shown below. Note that the siren output is defined as an INOUT signal indicating
that its value can be read from as well as written to, inside the model.

```
ENTITY auto_alarm IS
  PORT:
    SIGNAL sensor_bus : IN bit vector (3 DOWNTO 0) := (OTHERS => '0');
    SIGNAL position_sensor1 : IN bit := '0';
    SIGNAL position_sensor2 : IN bit := '0';
    SIGNAL key_sw : IN bit := '0';
    SIGNAL siren : INOUT bit);
END ENTITY auto_alarm;
```

Architecture Description

The model architecture has three process statements that are used to co-ordinate between
the key inputs and the sensor inputs and provide a warning alarm to the siren output if a se-
curity infringement occurs.

The first process statement sets the locally declared BOOLEAN signal `alarm` to TRUE if any
of the sensors indicate that there is a problem.

```
PROCESS(sensor_bus, position_sensor1, position_sensor2) BEGIN
  alarm <= sensor_bus(0) OR sensor_bus(1) OR sensor_bus(2) OR sensor_bus(3) OR
          position_sensor1 OR position_sensor2;
END PROCESS;
```

The second process sets the locally declared BOOLEAN signal `armed` to TRUE if the key
switch is turned on. Note that there is a 30 ms delay between the switching on of the key
switch and the arming of the system.

```
PROCESS BEGIN
  WAIT ON key_sw;
  WAIT FOR 30 ms;
  IF (key_sw = '1') THEN
    armed <= true;
  ELSE
    armed <= false;
  END IF;
END PROCESS;
```

The third process sets the siren on if the system is armed and an alarm has been detected by
the sensors. It incorporates a delay of 30 ms from the time of detection of an alarm in an armed
system to the time that a siren is sounded.

```
PROCESS BEGIN
  WAIT ON armed, alarm;
  IF armed AND (alarm = '1') THEN
    WAIT FOR 30 ms;
    siren <= '1';
  END IF;
END PROCESS;
```
The fourth process shuts off the siren if the key switch is turned off. Since the system requires 30 ms for key switch inputs to have an effect on the system (refer to the second process), it takes 30 ms for the siren to turn off after the key switch has been turned off.

Note that even though the statements within a process are sequentially executed, the four process statements themselves are concurrently executed.
Results

In this example scenario, the system behavior is described with the repeated enabling and disabling of the alarm system (using the key switch input pattern) and repeated security infringements (triggering the sensor inputs). The triggering of the sensor inputs may be caused by ajar doors, ajar trunks, breaking of glass, open windows, etc.

The system behavior is simulated for 1 sec, and the effect of the trigger inputs from the different sensors on the digital controller is described below.

Sensor Bus Trigger
Initially, the system is armed (at 30 ms), and then one of the sensor bus signals (sensor_bus[2]) is triggered (at 100 ms). This causes the siren to sound (at 130 ms). The siren is turned off when the key switch signal is disabled (at 250 ms).

Sensor 1 Input
After the system is armed again (at 340 ms), the position_sensor_1 input to the digital controller input from the simple_microswitch sensor model is triggered (at 450 ms). This causes the siren to sound for 100 ms even though the sensor trigger is reset in 50 ms. The siren is turned off when the key switch signal is disabled (at 550 ms).

Sensor 2 Input
The system is armed again (at 620 ms) and the position_sensor_2 input to the digital controller input from the advanced_microswitch sensor model is triggered (at 700 ms). However, the internal response time of the sensor model delays the sensor output (to 715 ms). The siren sounds (at 745 ms) till the key switch is disabled at 850 ms.

If the simple_microswitch sensor model is used instead of the advanced_microswitch sensor for the Sensor 2 Input, then the siren sounds earlier (at 730 ms) due to the absence of an internal delay.

![Diagram of sensor inputs and controller states](image-url)
VHDL-AMS Export

The standardization of the VHDL-AMS language ensures that models developed in VHDL-AMS are exchangeable among simulators from different vendors. Models created in SIMPLORER using VHDL-AMS can be exported to ASCII files for simulation in other VHDL-AMS tools. It is not only possible to export one or more VHDL-AMS models from the ModelAgent but to also export entire sheets with VHDL-AMS models. The export of models from the ModelAgent is discussed in “Adding VHDL-AMS Models to a Library” on page 150. Exporting sheets to VHDL-AMS netlists is demonstrated in this case study example.

Sheets can be built with VHDL-AMS models from model libraries, graphical subsheets, and text subsheets. These schematics can be exported to “pure” VHDL-AMS netlists that can be used in other simulation tools. Sheets that not only contain VHDL-AMS models but also internal/SML/C (non-VHDL-AMS) models can be exported to a VHDL-AMS netlist. In this case, the non-VHDL-AMS models are exported as foreign models, compliant with the IEEE 1076.1 standard.

Netlists containing foreign models can only be simulated in SIMPLORER and in most cases cannot be used with other VHDL-AMS simulation tools. To use these netlists in another simulator, the foreign models must be manually replaced in the code by equivalent foreign models that the target simulator can use.

The Automobile Alarm System case study sheet has the following types of models:
- VHDL-AMS models from multiple libraries
- VHDL-AMS text subsheet models
- Graphical subsheet model with SML sensor model and VHDL-AMS models
- Transformation models for signal-quantity conversions
- Stimulus generator model that generates VHDL code

Exporting a VHDL-AMS Model

To initiate the export of a Schematic sheet to a VHDL-AMS netlist:

1. Choose SHEET > EXPORT VHDL-AMS MODEL DESCRIPTION or FILE > EXPORT VHDL-AMS MODEL DESCRIPTION. The following dialog appears:

2. Enter the name of the VHDL-AMS file that will contain the exported code of the top-level entity. Click on [ ] to browse to a new location if necessary. This file will be given the .vhd extension.
A suggested name is shown in "Name of Top Level Entity", which specifies the top-level entity name in the exported VHDL-AMS netlist. The top-level entity instantiates the components on the sheet and provides the interconnection information between the components. The suggested name can be modified if necessary, but the name should conform to VHDL-AMS identifier syntax requirements (see "VHDL-AMS Language Fundamentals" on page 165 for more details). If the name does not conform to the VHDL-AMS identifier syntax requirements, a dialog box appears with the message "Please enter a valid name for the top level entity."

Apart from the top-level entity, it is also possible to export the VHDL-AMS code of the individual library models that are used in the sheet. For example, in this case study, the VHDL-AMS code for the step function block, summation block, position source models, and so on can be exported. To export individual models, choose either one of the following options from the «Components» frame:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>«Export all components»</td>
<td>Exports the source code of all the VHDL-AMS models on the Schematic.</td>
</tr>
<tr>
<td>«Export only user-defined components»</td>
<td>Exports the source code of only those models that are not available in the pre-installed SIMPLORER libraries.</td>
</tr>
</tbody>
</table>

When either of these options is chosen, the frame titled «Export Components to» becomes enabled. This frame allows the user to specify the target location of the source code belonging to the individual models on the sheet.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>«Same file as top-level entity»</td>
<td>Generates one VHDL-AMS file which contains the source code of all the models as well as the description of the top-level entity.</td>
</tr>
<tr>
<td>«Single file separate from top-level entity»</td>
<td>Generates the source code of the models in a file called file_name_Components.vhd where file_name refers to the name of the file provided earlier.</td>
</tr>
<tr>
<td>«Individual files separate from top-level entity»</td>
<td>Creates several VHDL-AMS files separate from the file containing the top-level netlist. Each of these files contains the VHDL-AMS description of a single model used in the sheet. The file name is derived from the name of the entity that the file contains.</td>
</tr>
</tbody>
</table>

A sheet can contain models that are not developed in VHDL-AMS, but are instead developed as internal/SML/C models. These models can also be exported to a netlist as foreign models with a VHDL-AMS wrapper. Foreign models generated by SIMPLORER can only be simulated in SIMPLORER. To allow VHDL-AMS wrappers to be created around non-VHDL-AMS models, select the «Export SML components» check box. If the sheet contains non-VHDL-AMS models and this check box is not selected, then the export operation will not be completed.
For this example, choose to export all components. Since the advanced microswitch model contains an SML component, check the box to export SML components. Choose to export all components to the same file as the top-level entity.

The Build Window shows messages from the export operation. The following code is generated for the top-level entity:

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
LIBRARY transformations;
USE transformations.omnicaster_package.ALL;
USE IEEE.mechanical_systems.ALL;
USE IEEE.math_real.ALL;
USE IEEE.fundamental_constants.ALL;
USE IEEE.electrical_systems.ALL;
LIBRARY STD;
LIBRARY basic_vhdlams;
ENTITY case_study_automotive_alarm_system IS
BEGIN
END ENTITY case_study_automotive_alarm_system;
```
ARCHITECTURE struct OF case_study_automotive_alarm_system IS

TERMINAL N0035 : TRANSLATIONAL;
QUANTITY Q_1 : VELOCITY := 0.0;
TERMINAL N0039 : TRANSLATIONAL;
QUANTITY Q_2 : VELOCITY := 0.0;
QUANTITY Q_3 : REAL := 0.0;
QUANTITY Q_4 : VOLTAGE := 0.0;
SIGNAL S_1 : bit := bit'LEFT;
SIGNAL S_2 : bit := bit'LEFT;
SIGNAL S_3 : bit := bit'LEFT;
SIGNAL S_4 : bit := bit'LEFT;
SIGNAL S_5 : bit := bit'LEFT;
SIGNAL S_6 : bit := bit'LEFT;
SIGNAL S_7 : bit := bit'LEFT;
SIGNAL S_8 : bit := bit'LEFT;
BEGIN
stimulus1: ENTITY WORK.stimulus1
  GENERIC MAP ( bvrsig1_val2_period => 5 , bvrsig1_val1_period => 5 , bvrsig1_len => 4 , stim_end => 1.000000e+000 , bc_low => 1.000000e-002 , bc_high => 1.000000e-002 )
  PORT MAP ( key => S_1 , bvrsig1(3) => S_4 , bvrsig1(2) => S_5 , bvrsig1(1) => S_6 , bvrsig1(0) => S_7 , clk => S_8 );
OmniCaster1: ENTITY transformations.realqty_bitsig(behav)
  GENERIC MAP ( thres => 5.000000e-001 )
  PORT MAP ( val => S_3 , inp => Q_3 );
OmniCaster2: ENTITY transformations.realqty_bitsig(behav)
  GENERIC MAP ( thres => 5.000000e-001 )
  PORT MAP ( val => S_2 , inp => Q_4 );
digital_controller: ENTITY WORK.auto_alarm
  PORT MAP ( key_sw => S_1 , position_sensor2 => S_2 , position_sensor1 => S_3 , sensor_bus(3) => S_4 , sensor_bus(2) => S_5 , sensor_bus(1) => S_6 , sensor_bus(0) => S_7 );
advanced_microswitch: ENTITY WORK.Macro1
  PORT MAP ( N0044 => N0039, ctrl => Q_4 );
step2 : ENTITY basic_vhdlams.step(behav)
  GENERIC MAP ( y0 => 0.000000e+000 , ts => 0.000000e+000 , t0 => 5.000000e-001 )
  PORT MAP ( val => Q_2 , ac_phase => 0.000000e+000 , ac_mag => 1.000000e-003 , ampl => 1.000000e+000 );
simple_microswitch1 : ENTITY WORK.simple_microswitch(behav)
  GENERIC MAP ( position_threshold => 5.000000e-004 )
  PORT MAP ( mech_input => N0035, control_output => Q_3 );
step1 : ENTITY basic_vhdlams.step(behav)
  GENERIC MAP ( y0 => 0.000000e+000 , ts => 0.000000e+000 , t0 => 4.500000e-001 )
  PORT MAP ( val => Q_1 , ac_phase => 0.000000e+000 , ac_mag => 1.000000e-003 , ampl => 1.000000e+000 );
vel_src2 : ENTITY basic_vhdlams.v_trb(behav)
  GENERIC MAP ( s0 => 0.000000e+000 )
  PORT MAP ( trb1 => N0039, ac_phase => 0.000000e+000 , ac_mag => 1.000000e-003 , value => Q_2 );
vel_src1 : ENTITY basic_vhdlams.v_trb(behav)
  GENERIC MAP ( s0 => 0.000000e+000)
  PORT MAP ( trb1 => N0035, ac_phase => 0.000000e+000 , ac_mag => 1.000000e-003 , value => Q_1 );
END ARCHITECTURE struct;
The exported netlist includes all the libraries and associated packages required for this example. The top-level entity has an empty interface since it represents the top-level sheet. The architecture for this entity declares the terminals, quantities and signals used for interconnections in this example. The architecture body instantiates the components used in the example.

**Foreign Models**

The SML sensor model used in the advanced microswitch subsheet can be exported as a foreign model. Foreign models contain instantiations of non-VHDL-AMS models within VHDL-AMS wrappers, and cannot be exchanged among simulators. The following figure shows the code snippet of the wrapper that is created around the sensor model.

```
-- ********** Foreign Model Wrapper Start **********
LIBRARY basic_vhdlams, digital_elements, iss, transformations;
USE basic_vhdlams.all, digital_elements.all, iss.all, transformations.all;
ENTITY sml_rampswitch_df_i11 IS
  PORT |
    TERMINAL sp : ELECTRICAL;
    TERMINAL stn : ELECTRICAL;
    TERMINAL ref : TRANSLATIONAL;
    TERMINAL position : TRANSLATIONAL;
END ENTITY sml_rampswitch_df_i11;
ARCHITECTURE sml_arch OF sml_rampswitch_df_i11 IS
  COMPONENT rampswitch_df_i11 IS
    PORT |
      TERMINAL sp : ELECTRICAL;
      TERMINAL stn : ELECTRICAL;
      TERMINAL ref : TRANSLATIONAL;
      TERMINAL position : TRANSLATIONAL;
  END COMPONENT;
brahim_c: COMPONENT rampswitch_df_i11 PORT sp, stn, ref, position;
ATTRIBUTES sml_model => STRING;
ATTRIBUTES sml_model => COMPONENT IS
  SML_RAMPSWITCH_DF_I11ramento: COMPONENT IS
  TERMINAL position ; port name in string'position' ; port name in string'position'
BEGIN
  sml_rampswitch_df_i11 PORT sp, stn, ref, position;
END COMPONENT;
brahim_c: COMPONENT rampswitch_df_i11 PORT sp, stn, ref, position;
END ARCHITECTURE sml_arch;
```

The foreign model wrapper specifies the interface of the sensor model in an entity description. The architecture description of the foreign model specifies a component declaration and instantiation for the sensor model. It also associates an attribute called `sml_model` to the component that specifies the SML call line of the sensor model.

Another example of the use of foreign models can be seen in the source code of the continuous memory block in the `basic_vhdlams` library (Blocks>Continuous>Memory). This model has two architectures called `behav` and `sml_behav`. The second architecture instantiates an SML model as a component and associates two attributes to the component. The attribute `sml_model` specifies the SML call line and the second attribute `sml_output` specifies output connections for the model.
5 Model Development

This chapter explains the use, development, export, and import of VHDL-AMS models in SIMPLORER. There are several ways to include standard or user-defined VHDL-AMS models in the Schematic environment to make them available for simulation.

A SIMPLORER simulation script (SML description) can consist of different model types. A model, used in an SML description can be an internal component, a C-model (modeled in C++), or a macro model. A macro itself is a substructure that can include all model types: internal components, C-models, and macros. There are two macro formats, graphical macros and text macros. The type of a model already placed on the sheet can be found in its Properties dialog in the «Library» tab. Each model type has a corresponding symbol in the ModelAgent. See also “Symbols in the ModelAgent” on page 140.

All models (internal components, C-models, macro models) belong to libraries. The various model types can be placed on the sheet and used in a simulation in the same manner.

This chapter contains information on:

- VHDL-AMS subsheets in Schematic
- Model libraries in SIMPLORER
- VHDL-AMS models in the ModelAgent
- Packages in the ModelAgent
5.1 VHDL-AMS Subsheets in Schematic

Subsheets allow the structuring of models and creation of hierarchical models. Models created from a subsheet definition can be used to add macros to model libraries in the ModelAgent. Macros can be used in any model description, as can basic models (drag-and-drop from the ModelAgent). Symbols can be developed for any subsheet. Choose «Edit Symbol» on the model shortcut menu to start the Symbol Editor and change the symbol. Schematic provides two types of subsheets: graphical and text.

Graphical and Text Subsheets

Graphical Subsheets

Graphical subsheets contain subsystems composed of models, just as sheets do. SML and VHDL-AMS models (or any simulateable models from the ModelAgent) can be placed on a graphical subsheet. SML and VHDL-AMS text subsheets can also be placed on a graphical subsheet. Graphical subsheets can contain additional graphical subsheets, and can be nested to any level.

Graphical subsheets can be created from a blank subsheet, or from models selected on an existing sheet. Graphical subsheets can be copied from a sheet and pasted directly into a library in the ModelAgent.

Text Subsheets

Text subsheets can be created as SML or VHDL-AMS descriptions. To create a new SML subsheet choose SHEET>SUBSHEET>NEW SML. To create a new VHDL-AMS subsheet, choose SHEET>SUBSHEET>NEW VHDL-AMS. If a text subsheet command is started, the Text Editor is opened as an embedded application. All functions such as search and replace, syntax coloring, and syntax checking are available within the Text Editor window.

Text models can be typed in manually, copied and pasted from another file, or imported directly into a text subsheet. To import a text model, right click in the text editor and select «Import». Navigate to the file (.vhd or .sml) to be imported, and click <Open>. The contents of the file will be copied into the sheet.

If a text model with multiple architectures is imported into a subsheet, only the last architecture will be available for use in that instantiation of the model. If multiple architectures are needed, import the model into a library and use it from the library.

Creating Graphical Subsheets

1. To start with a blank subsheet choose SHEET>SUBSHEET>NEW GRAPHICAL. To create a subsheet from a selection, select the models on the sheet (including the wiring) and choose SHEET>SUBSHEET>NEW GRAPHICAL FROM SELECTION.

2. Draw the subsheet symbol to the desired size by holding the left mouse button and dragging. A rectangular symbol is displayed, and the empty subsheet is opened.

3. Place and connect the models.
4 Create conservative nodes by selecting SHEET>SUBSHEET>CREATE PIN/PARAMETER. Conservative and non-conservative nodes serve as connectors from the substructure to the next higher model level and appear as pins on the model.

5 After creating the nodes, connect the pin symbols to the appropriate places in the circuit.

6 When the subsheet is finished, right-click and select «Level Up» to return to the sheet.

7 Right-click on the subsheet and select «Edit Symbol» to create a symbol for the subsheet.

Creating Text Subsheets in VHDL-AMS

1 Choose SHEET>SUBSHEET>NEW VHDL-AMS. The mouse pointer becomes a cross wire.

2 Draw the subsheet symbol to the desired size by holding the left mouse button and dragging. A rectangular symbol is displayed, and the empty text editor is opened.

3 Enter the subsheet description in VHDL-AMS. Type in the code, or right click and import the code from a text file.

This example explains the modeling of a capacitor, for which the capacitance is input to the model as a parameter.

The entity defines two electrical terminals that represent plus (+) and minus (–) pins of the model. The parameter for capacitance is a value of data type REAL defined through a GENERIC statement. This parameter value is constant during the simulation but may be varied between simulations. Also, the parameter value can be provided as an expression that is evaluated once at the beginning of the simulation.

The architecture defines the constant value of the capacitor as data type REAL of 1µF. The capacitance value is used to calculate current as in the following equation:

\[ \text{current} = \text{capacitance} \times \frac{\delta \text{voltage}}{\delta t}. \]

The statement

```
  current == capacitance \times \text{voltage}'\text{DOT};
```

uses the ‘\text{DOT}’ attribute on the analog voltage quantity to obtain the first derivative of voltage with respect to time.

```
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
ENTITY ParameterCapacitor IS
  GENERIC (capacitance : REAL := 1.0e-6);
PORT (TERMINAL p,m : ELECTRICAL);
END ENTITY ParameterCapacitor;
ARCHITECTURE behav OF ParameterCapacitor IS
  QUANTITY voltage ACROSS current THROUGH p TO m;
BEGIN
  current == capacitance \times \text{voltage}'\text{DOT};
END behav;
```

4 To check the syntax, click the Check symbol on the toolbar. If the compiler detects an error, a message box appears on the screen.
Choose «Level Up» on the Text Editor’s shortcut menu (right mouse click in the editor window) to switch to the main sheet. Double-click the subsheet symbol to open the subsheet Properties dialog. In the «Output/Display» dialog, the defined terminals are listed. To change the pin position at the symbol, activate the move pin mode using CONNECT > MOVE PIN, and drag the pin with pressed mouse button to a new position.

### Copying Subsheets

Subsheets can be copied (CTRL+C) and inserted (CTRL+V) in the same or another sheet. The subsheets can be edited after they are inserted. Copied subsheets are independent of one another, and will retain the edits made in the individual sheets.

### Duplicating Subsheets

Subsheets can also be duplicated (EDIT > DUPLICATE or CTRL+D). All subsheets created with the duplicate feature will always be exactly alike. Any changes made to a duplicated subsheet are propagated to all corresponding duplicate subsheets.

### Adding Subsheets to a Library

In a simulation model, defined subsheets (SML/VHDL-AMS text or graphical) can then be added to model libraries and used in other sheets. To do so, insert subsheets from a model description or paste the copied subsheet directly into a model library.
5.2 Library and Model Management in the ModelAgent

SIMPLORER stores models in model libraries. The ModelAgent manages and maintains all libraries and models and provides editing functions. The ModelAgent can appear as an embedded pane in Schematic and Symbol Editor, or can be used as a separate SIMPLORER application. To start the ModelAgent from within the SSC Commander, double-click the ModelAgent symbol , or choose PROGRAMS >> MODELAGENT.

ModelAgent Working Window

When first starting the ModelAgent, a window similar to the following appears. Model trees in the side panes contain the installed libraries along with their corresponding models. Installed VHDL-AMS models appear on the «AMS», «Digital», and «Tools» tabs.

If different VIEW options are selected, the window may look different from the figure shown above. To get the same display as shown, enable all options on the VIEW menu and then maximize the window.
Two windows display the structure of the installed libraries and the models included in each of them. When a library is selected, its model tree appears in the lower-left pane. If the mouse pointer is placed over the model name, a tool tip appears displaying its symbol.

The working window displays the parameters and connections for the selected model. From this window, the model text can be created or modified. To change a model’s symbol, select the model and press the <Edit Symbol> button to open the Symbol Editor.

**Model Libraries and Models**

To use a model in SIMPLORER, it must be included in a library, and the library must be available in the ModelAgent. Use the FILE menu commands to install and configure the libraries in the SIMPLORER program environment. Use the ELEMENT menu commands to arrange the models within a library. User-defined libraries can contain both user-defined models as well as models copied from installed libraries.

**Managing Model Libraries**

All model libraries used in SIMPLORER are .smd (SIMPLORER Model Database) files. Each library must be included in the SIMPLORER environment.

Libraries can be created, added, and removed from the following locations:

- ModelAgent FILE menu
- Model tree shortcut menu (in ModelAgent, Schematic, and Symbol Editor)
- SSC Commander OPTIONS>MODEL DATABASE (no library creation available)

Select the tab («Basics», «Displays», «AMS», «Digital», «Tools», «Add Ons», «Manufacturers», «Users», «Projects») where the libraries should be included. Libraries assigned to the «Projects» tab are available only for the corresponding project. To use the library in other projects, include it again in that project or assign it to other tabs.

Most of the commands described in this chapter are available only if a library is unlocked (FILE>LOCK DATABASE) and if the model library file is writable. (Files that are sent by e-mail may be write-protected.)

**File Menu**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT LIBRARY</td>
<td>Inserts an existing library (.smd) into the model tree. If a 6.0 library is inserted, a prompt to update the library to Version 7.0 format appears.</td>
</tr>
<tr>
<td>NEW LIBRARY</td>
<td>Creates a new model library. Select the language resources and create a new library by choosing &lt;OK&gt;. See also “Creating Model Libraries” on page 141.</td>
</tr>
<tr>
<td>REMOVE LIBRARY</td>
<td>Removes a library from the model tree (but does not erase it from the hard disk).</td>
</tr>
<tr>
<td>SIMPLORER MODELING IMPORT</td>
<td>Imports .smu model files from the SIMPLORER Web Database.</td>
</tr>
</tbody>
</table>
### Command Description

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE INTERFACE DESCRIPTION</td>
<td>Creates a text description of all models used in the selected library and their parameters, exporting a table containing the following fields: Name, Parameter, Direction, Dynamic/Vector, Output, Default, Unit, Data Type, Node, Nature, &quot;Language Resources&quot;, Path in library.</td>
</tr>
<tr>
<td>COMPILATE ALL TEXT MODELS</td>
<td>Compiles all VHDL-AMS models in the selected library. This process is needed when model descriptions/packages were changed.</td>
</tr>
<tr>
<td>COMPACT AND REPAIR CURRENT LIBRARY…</td>
<td>Compacts and repairs the selected model library. Before starting the repair process, close other programs (Schematic, Symbol Editor) that use this model library.</td>
</tr>
<tr>
<td>COMPACT AND REPAIR LIBRARY…</td>
<td>Compacts and repairs any SIMPLORER model library. Enter the name of the library, and start the process. Before starting the repair process, uninstall the corresponding model library.</td>
</tr>
<tr>
<td>CONVERT 4.x LIBRARY</td>
<td>Converts an existing SIMPLORER 4.x library into a SIMPLORER 7.0 library.</td>
</tr>
<tr>
<td>EDIT STRINGS</td>
<td>Displays all text strings used in the selected library, and allows the entries to be modified. The order of the strings is irrelevant to the models.</td>
</tr>
<tr>
<td>EXPORT LIBRARY</td>
<td>Creates a separate .sml or .vhd file from the selected model library with all model descriptions, either in SML text (choose SML20) or VHDL-AMS (choose VHDL-AMS).</td>
</tr>
<tr>
<td>LOCK DATABASE</td>
<td>Locks the selected library in the model tree. Changes cannot be made to a library if it is locked.</td>
</tr>
<tr>
<td>PROPERTIES</td>
<td>Displays all existing language tables from the selected library (text and modeling language). Add a new resource with &lt;New&gt;. See also “SIMPLORER Language Support” on page 139.</td>
</tr>
<tr>
<td>EXIT</td>
<td>Closes the ModelAgent.</td>
</tr>
</tbody>
</table>

Content folders can be created to structure the content within a library. These content folders make it easier to search in large libraries. Use the ELEMENTS > NEW > FOLDER menu command to create a new folder. The name of the folder can be changed immediately, or it can be done later by clicking twice in the name field and entering a new name.
**Model Development**

**Managing Models**

SIMPLORER can be used to create user-defined models. To use a model in SIMPLORER, it must be included in a library, and the library must be available in the ModelAgent. Use the **ELEMENT** menu commands to create, insert, modify, and arrange models within a library.

Most of the commands described in this chapter are available only if a model is unlocked (**ELEMENTS > LOCK ELEMENT**) and if the model library file is writable. (Files that are sent by e-mail may be write-protected). Models from the installed libraries cannot be edited directly. To change an installed model, copy it to a user library such as **<Users>** and edit that model.

Models can be reorganized within the libraries as follows:

- **Copy**
  Hold down the CTRL key and drag the model to a new location within the model tree or choose **ELEMENTS > COPY**.

- **Move**
  Drag the model to a new location within the model tree or choose **ELEMENTS > PASTE**, **ELEMENTS > PASTE ABOVE**, or **ELEMENTS > PASTE IN FOLDER**. A message confirming the move appears, and gives the option of suppressing the message in the future.

- **Delete**
  Choose **ELEMENTS > REMOVE ELEMENT** and click <Yes> in the dialog box that appears.

- **If models are deleted from the installed libraries, then SIMPLORER needs to be reinstalled to restore these models.**

### Element Menu ▼

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NEW</strong></td>
<td>Creates a new folder or macro, or imports a C-model from a .dll file.</td>
</tr>
<tr>
<td></td>
<td>See also &quot;Creating Models&quot; on page 142.</td>
</tr>
<tr>
<td><strong>INSERT</strong></td>
<td>Imports SML or VHDL-AMS macros from ASCII files.</td>
</tr>
<tr>
<td><strong>EXPORT</strong></td>
<td>Creates a separate .sml or .vhd file from the selected model, either in SML</td>
</tr>
<tr>
<td></td>
<td>text (choose SML20 MACRO) or VHDL-AMS (choose VHDL-AMS MACRO).</td>
</tr>
<tr>
<td><strong>REMOVE ELEMENT</strong></td>
<td>Permanently deletes the selected model. The model cannot be restored.</td>
</tr>
<tr>
<td><strong>UPDATE C-MODEL</strong></td>
<td>Starts an update for the selected C-model.</td>
</tr>
<tr>
<td><strong>EDIT SYMBOL</strong></td>
<td>Opens the symbol editor to modify the symbol for element.</td>
</tr>
<tr>
<td><strong>EDIT SUBSHEET...</strong></td>
<td>For a subsheet model, opens the subsheet for editing.</td>
</tr>
<tr>
<td><strong>EDIT COMPONENT</strong></td>
<td>Launches the Component Dialog Wizard to create a dialog to be used</td>
</tr>
<tr>
<td><strong>DIALOG</strong></td>
<td>to input model parameters.</td>
</tr>
<tr>
<td><strong>COPY</strong></td>
<td>Copies the selected model to a specified new location.</td>
</tr>
<tr>
<td><strong>PASTE</strong></td>
<td>Inserts the model below the selected model in the tree.</td>
</tr>
<tr>
<td><strong>PASTE ABOVE</strong></td>
<td>Inserts the model above the selected model in the tree.</td>
</tr>
</tbody>
</table>
SIMPLORER Language Support

SIMPLORER supports different languages for the following items:
- Program menus and dialogs.
- Model libraries.

The language for program menus and dialogs is defined in the SSC Commander, via OPTIONS>LANGUAGE. Select «German (Germany)» or «English (United States)», and click <OK>. After an additional dialog box, SIMPLORER closes and restarts with the new language for menus and dialogs. All open SIMPLORER files need to be saved and closed before initiating the process of changing the language.

The language for model libraries is defined when a new library is created or with FILE>PROPERTIES in the ModelAgent for an existing library.

To add a language resource to an existing library, do the following:

1. Select the required library. Choose FILE>PROPERTIES in the ModelAgent or choose «Properties» on the shortcut menu of a tab in the ModelAgent, Schematic, or Symbol Editor.
2. Click <New> in the «Resource Tables» area, and select a language for model text strings. Parameter names are identical for all language resources.
   a. Select «Copy Resource» to copy data from an existing language.
   b. Select one of the existing languages from the list.
   c. Select Symbols, Keywords, Infos, and/or Files settings, and then click <OK>.
3. Click <OK>. A new set of entries appears in the list.
4. Click <OK> to save the settings.

The new language resource is available for all models in the library. If a language’s resources are not copied, no entry is made in the title field. For the other values, default settings are created by the ModelAgent.

Set library languages from the following locations:
- SSC Commander, via OPTIONS>LANGUAGE «Library Languages» (for model names displayed in the model tree).
- Single Schematic sheet, via SHEET>PROPERTIES «System» (for names and text used in models placed on the sheet). The option «System dependent» in this dialog means the library language defined in the SSC Commander is used.

**Command** | **Description**
---|---
PASTE IN FOLDER | Inserts the model in the selected folder in the tree.
HELP FOR ELEMENT | Opens the help file for the selected model, if available.
EXAMPLE | Opens the example file for the selected model, if available.
LOCK ELEMENT | Locks the selected model. If a model is locked, no more changes can be made. To remove the lock, copy the model to a different library or to another location in the same library.
PROPERTIES | Opens the input dialog to edit the macro content.
The following table shows available language settings for menus and dialogs and for model libraries and the effect on SIMPLORER:

<table>
<thead>
<tr>
<th>Settings</th>
<th>Menu and Dialogs</th>
<th>Model Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSC Commander: OPTIONS&gt;LANGUAGE</td>
<td>SSC Commander: OPTIONS&gt;LANGUAGE</td>
<td>Schematic: SHEET&gt;PROPERTIES</td>
</tr>
<tr>
<td>«SIMPLORER Language»</td>
<td>«Library Language»</td>
<td>«System» tab</td>
</tr>
<tr>
<td>(German or English)</td>
<td></td>
<td>«Language»</td>
</tr>
<tr>
<td>Effect on</td>
<td>Program menus and text in dialog of</td>
<td>Names in Model Trees</td>
</tr>
<tr>
<td></td>
<td>all SIMPLORER programs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Names and strings in a single Schematic sheet</td>
<td></td>
</tr>
</tbody>
</table>

Symbols in the ModelAgent

- Green folder. Folder containing packages used from VHDL-AMS models.
- Green rectangle. VHDL-AMS packages definitions. The element cannot be placed on the Schematic.
- Yellow folder. Folder containing models that can be placed on the sheet and used for simulation.
- Green dot circle. Internal SIMPLORER models. Most of the models available in the «Basics» tab of the ModelAgent are internal models. These components are evaluated within SIMPLORER's internal simulator.
- Blue dot circle. Standard and user-defined C-models. There are a few models of this type in the «Basics» tab. These models, available from an external C++ DLL are developed using the C-Model Interface.
- Yellow rectangle. Standard and user-defined text macros. Macros can be created, edited, and encoded in the ModelAgent in SML or VHDL-AMS text. An encoded macro script cannot be edited again. A text macro on the sheet cannot be opened.
- Light-blue rectangle. Standard and user-defined graphical macros. Macros can be created in Schematic from a graphical selection. A graphical macro (sub-sheet) on the sheet can be opened. Graphical macros can be created in the ModelAgent by pasting graphical subsheets copied from Schematic.
- Green rhombus. Coupling models. The models are calculated by different simulators.
- Red triangle. Elements for displaying simulation results directly on the sheet in Schematic. Results of these elements cannot be used as parameters for simulation.
- Blue padlock added to a symbol. Folders and models are locked and possibly write protected and cannot be changed in the ModelAgent. Copy the folder and/or model to a new location to remove the lock.
- Light-blue padlock added to a symbol. Folders and models are write protected and cannot be changed in the ModelAgent. Change the attribute of the .smd file in Windows explorer to remove the write protection.
Model Symbols

Every model in a library has at least one symbol, which is displayed when the model is dropped on the Schematic sheet. Model symbols are language-dependent, which means that each language can specify its own symbol.

Existing model symbols, such as the default symbol of a user-defined macro or the symbol of a SIMPLORER standard model, can be modified using the Symbol Editor.

When a model is created, ModelAgent assigns it a symbol that is used by Schematic to display the model on the sheet. This symbol can be changed in the Symbol Editor by selecting the menu ELEMENTS -> EDIT SYMBOL and making the desired change.

The automatically created default symbol for user-defined models (macros, C-models) is a simple rectangle with pins for conservative nodes defined in the corresponding model description, and an associated text element containing the name of the model. Pins for non-conservative nodes can be displayed using the display options that are available in the model’s Properties dialog.

5.3 Managing VHDL-AMS Models in the ModelAgent

The ModelAgent is used to manage and maintain all libraries and models, and provides access to model editing functions. In contrast to Schematic, where VHDL-AMS models defined directly on a sheet can only be used on that or other open sheets, the ModelAgent contains model definitions within model libraries that can be used on any model sheets.

To start the ModelAgent from within the SSC Commander, double-click the ModelAgent symbol, or choose PROGRAMS -> MODELAGENT.

Creating Model Libraries

To create a new library that can include VHDL-AMS models, do the following:

1. Choose FILE -> NEW LIBRARY in the ModelAgent, or choose «New Library» on the shortcut menu of a tab in ModelAgent, Schematic, or Symbol Editor. A file dialog appears.
2. Browse to the folder where the library file is to be saved and enter a name. Click «Save».
3. Click «New» in the «Resource Tables» area, and select a language for model text strings. Parameter names are identical for all language resources.
4. Click «OK». A new set of entries appears in the list.
5. Repeat steps 3 and 4 if more than one language is required.
6. Click «New» in the «Simulation Tables» area, and select VHDLA for the model description language. VHDLA will also allow SML models to be included in the library.
7. Click «OK». A new set of entries appears in the list (including SML access format table).
8. Click «OK» to create the new model library.
The model library is inserted into the model tree without a model. Models can be inserted by selecting ELEMENT>NEW, ELEMENT>INSERT, or ELEMENT>PASTE, or by dragging and dropping models from other libraries. The name for the model library and language resources can be modified later.

Creating Models

User-defined VHDL-AMS models can be added to user-defined libraries. There are three ways to create a new model:

• Directly in ModelAgent
• From an existing file
• Copying and pasting the model from another model library

Creating Models in VHDL-AMS in the ModelAgent

To create new VHDL-AMS macro models, a library with VHDL-AMS language tables is required. The macro model definition can be classified into two parts: an entity (interface) description and an architecture (behavior) description. The modeling process is illustrated below using a constant capacitor example. In this example, the capacitance value is defined as a constant in the architecture of the model; consequently, different architectures are used to provide different capacitance values.

The example explains the modeling of a constant capacitor with capacitances of 1mF and 10mF.

The entity defines two electrical terminals that represent plus (+) and minus (−) pins of the model. Two architectures are defined for the model: the first with a capacitance value of 1mF and the second with a capacitance value of 10mF. The capacitance value is used to calculate current as in the following equation:

\[
\text{current} = \text{capacitance} \times \frac{\text{voltage}}{\text{delta t}}.
\]

The equation

\[
\text{current} == \text{capacitance} \times \text{voltage}'\text{DOT};
\]

uses the 'DOT' attribute on the analog voltage quantity to obtain the first derivative of voltage with respect to time.

```vhdl
LIBRARY IEEE;
USE IEEE.ELECTRICALSYSTEMS.ALL;

ENTITY ConstantCapacitor IS
 PORT (p,m : ELECTRICAL);
END ENTITY ConstantCapacitor;

ARCHITECTURE c1mf OF ConstantCapacitor IS
   CONSTANT capacitance : CAPACITANCE := .001;
   QUANTITY voltage ACROSS current THROUGH p TO m;
BEGIN
   current == capacitance * voltage'DOT;
END c1mf;

ARCHITECTURE c10mf OF ConstantCapacitor IS
   CONSTANT capacitance : CAPACITANCE := .01;
   QUANTITY voltage ACROSS current THROUGH p TO m;
BEGIN
   current == capacitance * voltage'DOT;
END c10mf;
```
Creating a VHDL-AMS Model Manually

1. Select a library containing VHDL-AMS language tables (for example, the user library). This type of library has a green package folder at the top of the model tree. If model folders already exist, click the folder in which the new macro is to be inserted.

   If VHDL-AMS resources need to be added to a library, open the Properties dialog for the library, go to the «Properties Model Database» tab, and click the <New> button associated with the Simulation tables to add VHDLA.

2. Choose ELEMENTS>NEW>TEXT MODEL in the ModelAgent or choose «New» «Text Model...» on the shortcut menu of the model tree area in the ModelAgent. The Edit Element dialog opens.

3. Select VHDLA as the modeling language in the dialog.

4. Click the «General» tab and enter a title (used as name in the model tree).

5. Click the «Model Text» tab and check the «Enable» box.

6. Enter the model entity name and click <OK>. A template with the specified entity name appears in the dialog.

   - Check the «Enable» option
   - Enter the entity name
   - Automatically added entity template
Model Development

7. Enter the complete model interface description of the model in VHDL-AMS syntax in the «Entity» tab. Text can be added in the information line, if desired.

![Entity Diagram]

- Enter the complete entity description
- Enter the port information

8. Click the Add symbol on the upper right side to create a new «Architecture» tab. The dialog to define the architecture name appears.

9. Enter an architecture name in the dialog. A template with the specified architecture name appears in the Properties dialog.

![Architecture Dialog]

- Click the Add symbol
- Enter the architecture name
- Automatically added architecture body template

10. Enter the complete architecture description of the model into the «Architecture» tab.

![Architecture Code]

- Enter the complete architecture description
- Enter the info line
Several architectures can be created for an entity definition. The number of architectures is limited only by computer resources. In the example, the second architecture contains the definition for a 10mF capacitor.

Either of the two architectures specified for the model can be selected after the model is dropped on a sheet. The information line is displayed in parenthesis after the architecture name.

If all model definitions are finished, click <OK>.

It is possible to define the architecture that will be used for a model by loading configuration files. See also Chapter 3.5 Step 5: “Configuration Modes” on page 50.
Creating a VHDL-AMS Model Using the VHDL-AMS Wizard

Follow the first six steps in the section, “Creating a VHDL-AMS Model Manually” on page 143.

1. In the Edit Element dialog, press the Start Wizard button . The VHDL-AMS Wizard opens.

2. Open the ConstantCapacitor in the model tree. Right-click the Entity folder and choose «Edit Libraries and Packages». Check ieee under «Add Libraries» and ieee.electrical_systems under «Add Packages». Click <OK>.
3 Open the Entity folder in the model tree. Right-click the Port folder and choose «Edit Entity». Use the Add Port button to add two ports. They will by default be TERMINAL objects of type electrical.

4 Enter p for the first terminal name and m for the second terminal name. Enter a Comment if desired. Click <OK>.

5 Right-click on the Architecture folder in the model tree and select «Add Architecture». In the Add Architecture dialog, add two entries and name them c1mf and c10mf. Click <OK>.

6 Open the Architecture folder in the model tree. Open c1mf and right-click the Declarations folder. Choose «Edit Architecture Declaration». Add a Declaration, which will by default be a CONSTANT object. Enter capacitance as the name and select «capacitance» as the data type. Enter 1m as the value.
7. Add a new entry under Branches. Enter voltage as the Across aspect, enter current as the Through aspect, select a as the From Terminal and m as the To Terminal. Click <OK>.

8. Right-click c1mf and choose «Edit Architecture Body». Create the following equation in the dialog by double-clicking the equation components from the appropriate window areas. Model quantities can be found in the model tree, the double == (Equation) from the Simultaneous Statements template, mathematical operators from the Operators template, and ^ DOT from the Attributes template. Finish the equation with a semicolon and click <OK>.

9. Repeat steps 6 through 8 for c10mf, using 10m as the value for the capacitance.
Symbol Functions and Options in the Model Text Tab

In the «Model Text» tab there are some symbols and options with several functions to modify a model description. The symbols and their functions are listed in the following table:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Available In</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Wizard symbol" /></td>
<td>Wizard symbol. Starts the VHDL-AMS wizard.</td>
<td>Entity and Architecture tabs</td>
</tr>
<tr>
<td><img src="image2.png" alt="Add symbol" /></td>
<td>Add symbol. Creates a new architecture for an entity.</td>
<td>Entity and Architecture tabs</td>
</tr>
<tr>
<td><img src="image3.png" alt="Delete symbol" /></td>
<td>Delete symbol. Deletes the current architecture description.</td>
<td>Architecture tab</td>
</tr>
<tr>
<td><img src="image4.png" alt="Check symbol" /></td>
<td>Check symbol. Compiles the current VHDL-AMS description and displays messages in the field below the description.</td>
<td>Entity and Architecture tabs</td>
</tr>
<tr>
<td><img src="image5.png" alt="Encode button" /></td>
<td>Encode button. Encodes the description of the current Architecture description.</td>
<td>Architecture tab</td>
</tr>
<tr>
<td><img src="image6.png" alt="Enable option" /></td>
<td>Enable option. If the enable option is cleared, the complete model description is deleted.</td>
<td>Entity and Architecture tabs</td>
</tr>
<tr>
<td><img src="image7.png" alt="Default architecture description option" /></td>
<td>Default architecture description option. If the option is checked, the description is used as default when the model is dropped on a sheet in Schematic. Only one architecture can be defined as default.</td>
<td>Architecture tab</td>
</tr>
</tbody>
</table>

Modifying the Model Symbol

For a new model, a default symbol is automatically created. The symbol is a simple rectangle with pins for conservative nodes. To modify the symbol, do the following:

1. Select the model in the model tree and choose ELEMENTS > EDIT SYMBOL in the ModelAgent or choose «Edit Symbol» on the model’s shortcut menu in the ModelAgent or Schematic. The Symbol Editor opens showing the symbol of the selected model.
2 Modify the symbol with the edit functions.

3 Choose FILE > SAVE AND GO BACK to save the changes and close the Symbol Editor. The new symbol is used when the model is placed onto the sheet.

Adding VHDL-AMS Models to a Library

VHDL-AMS model files obtained from other sources can be added to a library. Additionally, VHDL-AMS models can be exported from an existing library and imported into a user library where they can be modified. The following example shows how to export a multiple architecture model and import it into a user library.

1 If the library used for the system examples and case studies is loaded, click on the «Projects» tab in the ModelAgent.

2 Right-click on the Load model, and choose «Export» «VHDLA Models» on the shortcut menu. Navigate to the location where the file will be stored (in this case choose the Tutorial Examples folder), and give the file a name. The extension .vhd will be added by default.

3 In the ModelAgent, choose the tab of the library where the model will be stored (it must be a user library that is not locked).

4 Right-click in the ModelAgent window, and choose «Insert» «Model(s) from VHDL-AMS file» on the shortcut menu.
5 Navigate to the model file that was just saved and click «Open». The Insert Model dialog allows the importation of any or all of the architectures associated with the model.

6 Click <OK> to insert the model definitions in the selected library.

7 Right click on the newly inserted model and choose «Edit Symbol» on the shortcut menu to start the Symbol Editor, which can be used to modify the default symbol.

The model is complete now and is integrated in the selected library, and can be changed at any time. The model can now be dragged from the library onto any Schematic sheet and have all of its architectures available for use.

Model Properties

The ELEMENT>PROPERTIES command opens the Properties dialog of a selected model. The model shortcut menu can also be used to open the dialog.

All values available in these dialogs are only for the selected documentation language and modeling language.

General Properties Tab

The «General» tab contains general model information. The ModelAgent uses the title to display models in the model tree. Keywords are used to find models in search mode. The entries
in «Modified by» and «Description» have no function in other applications.

The name in «Modified by» is always replaced with active user name when the ModelAgent is closed.

Click <Assign> to display all keywords used in the library and assign them to the active model. If additional keywords other than those already in the library are needed, click twice in the Keywords field on the «General» tab and enter the new keyword. The <Assign> button is only active if keywords exist in the library that are not already assigned to the current model.

### Files Tab

The «Files» tab contains links to all associated files, including language-dependent files (upper list box) and language-independent files (lower list box). The language-independent files are used when no associated file is indicated for the active language setting.
Files can be added, removed, modified, and tested using the buttons next to the corresponding list box. Define the name and type of the file and, if required, a model ID. The files have the following functions:

<table>
<thead>
<tr>
<th>File Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>Associates a file describing how to use the model with the model. All file formats can be included. Clicking &lt;Load Example&gt; in the ModelAgent Working window opens all assigned files of this type. If relative paths are used, make sure that the SIMPLORER example directory is configured correctly.</td>
</tr>
<tr>
<td>WinHelp File</td>
<td>Defines a model help file in WinHelp format (.hlp file) or HTML Help format (.chm file). Only one file for each help format can be assigned. Both help files are started when a link is provided. If relative paths are used, make sure that the SIMPLORER help directory is configured correctly.</td>
</tr>
<tr>
<td>HTML Help File</td>
<td></td>
</tr>
<tr>
<td>Model Bar Bitmap</td>
<td>Defines a .bmp file 16x16 pixel used for the Model bar in the Schematic.</td>
</tr>
<tr>
<td>C Model</td>
<td>Defines the .dll file used by the model. (Only for C-models.)</td>
</tr>
<tr>
<td>Include in Library</td>
<td>If the box is checked for a file assignment, the entire file is included in the model database; otherwise only the link is included. Embedding files in the model library increases file size, but the library (.smd file) can be easily exchanged if all files are included.</td>
</tr>
</tbody>
</table>

**Simulation Tab**

The «Simulation» tab contains the call lines that will be included in the simulation script. No changes can be made in this dialog. Click <Edit> to start the Symbol Editor and modify the model symbol. See also “Symbol Animation” on page 101.

**Conservative Nodes Tab**

The «Conservative Nodes» tab lists the connections (nodes) that are defined in the model text, C-model, or internal component. A description and the default display setting can be entered for any nodes listed. Click the field in the respective column and enter the text or make a choice from the drop-down menu. See also “Across and Through Quantities of Natures” on page 12.

**Non-Conservative Nodes Tab**

The «Non-Conservative Nodes» tab (see following figure) lists the parameters defined in the model text. For each parameter, the Type option at the bottom of the tab can be used to specify
whether a pin is displayed by default for a parameter, and how it is displayed. Check «Use Pin» to turn on the display of the pin, then select an option box (Characteristic, External File, Logical Expression, Value, Control Signal, Control Quantity, Initial Value, Text) to determine how the pin will be displayed. The Types have no influence on the parameter itself.

The following values can be entered for each parameter, which will be used for each instance of the model placed on a sheet:

- Physical Type and Unit: physical type for the parameter and default unit (if not predefined)
- Preferred Unit: unit used in model dialogs
- Description: description of the parameter in the model Properties dialog
- Value: value of the parameter (will override any pre-defined value)
- Display: how the parameter should be displayed

To define an entry, click the field in the respective column, and enter the text. The default value must be a numerical value with or without a scaling suffix. The defined unit is used in model dialogs. These units are not used in Display Elements.

Each parameter in the list has a shortcut menu, where its default output settings can be defined. Select one of the following output formats for a selected parameter: Online or Result Database. If «Default Output for Probe» is checked, the parameter is displayed when the probe is placed on the sheet in Schematic. Only one parameter can be selected as the default for the probe.

To add a probe Display Element to a sheet, right-click a model on the sheet and select «Probe».
Model Text Tab

The «Model Text» tab displays the SML or VHDL-AMS description of the model. The model text can be edited directly here. Clicking <OK> checks the syntax of the text. Error messages are displayed below the input field in the message window. Internal models, C-models, and encoded macro models have no text in this dialog box.

If the macro text is encoded by clicking the <Encode> button in the dialog, the text can no longer be changed or viewed. This is in contrast to locked models, in which the entries in the text field can be viewed but not changed. For encoded macros, the message “The model text is encoded” is displayed in place of the text.

Tolerance Tab

The «Tolerance» tab allows the specification of tolerance values for parameters that can be used for multi-simulation tasks belonging to advanced analysis types, such as Monte Carlo Analysis. The distributions that can be used for specifying the tolerance include Uniform, Normal, Lognormal, and User-defined distributions. A user-defined distribution can be specified using a table defined in the SIMPLORER mdx format. The boundaries for the distribution can be specified by providing the mean value and either the standard deviation or the percentage tolerance.
Note that the tolerance parameters specified for this model will be applied to all new instances of this model on the Schematic. To set the tolerance parameters for a specific model instance, edit the model properties of that specific instance.

Parameter Sets Tab

The «Parameter Sets» tab allows multiple sets of parameters to be associated with each model. Each parameter set consists of a name for that parameter set, and the parameter values for the set. The parameter sets will appear under the model in the ModelAgent, and can be dragged onto the sheet to create an instance of the model with a specific parameter set.

Note that while only a model instance’s parameter set is visible on the Schematic sheet, all parameter sets associated with a model are visible in the ModelAgent.
Stress Tab

The «Stress» tab allows the specification of parameters for stress analysis tasks in SIMPLORER.
5.4 Managing VHDL-AMS Packages in the ModelAgent

Packages are collections of reusable declarations and definitions such as types, constants, functions, procedures, and nature. Packages have a green folder symbol in the tree and a green rectangle in front of their name. There are standardized packages from IEEE such as STD_LOGIC and TEXTIO, and proposed packages from IEEE such as ELECTRICAL_SYSTEMS and THERMAL_SYSTEMS. User-defined packages can be added to a model library and used in a model description.

This chapter describes VHDL-AMS packages and how they can be created and inserted into SIMPLORER.

Creating Packages in the ModelAgent

To create new VHDL-AMS packages, a library with VHDL-AMS language resources is required. The package definition can be classified into two parts, the package declaration and the package body definition. A simple package that performs integer-octal conversions is used to illustrate the use of packages in SIMPLORER.

The `octal_conversions` package converts an integer number to its octal equivalent and vice versa. The package defines an octal digit data type that has valid values between 0 and 7 and an octal number data type which is an array of octal digits. The package also declares two functions `int2oct` and `oct2int`.

The `int2oct` function converts an integer number to an octal number through the successive division method. The `oct2int` function converts an octal number to an integer number by raising each digit to the corresponding power of eight.

```vhdl
PACKAGE octal_conversions IS
    TYPE octal_digit IS RANGE 0 TO 7;
    TYPE octal_number IS ARRAY (NATURAL RANGE <>) OF octal_digit;
    FUNCTION int2oct(int : INTEGER; octal_length : INTEGER) RETURN octal_number;
    FUNCTION oct2int(oct : octal_number; octal_length : INTEGER) RETURN INTEGER;
END PACKAGE octal_conversions;

PACKAGE BODY octal_conversions IS
    FUNCTION int2oct(int : INTEGER; octal_length : INTEGER) RETURN octal_number IS
        VARIABLE temp_int : INTEGER := int;
        VARIABLE temp_oct : octal_number(octal_length-1 DOWNTO 0);
        BEGIN
            FOR j IN 0 TO (temp_oct'LENGTH-1) LOOP
                CASE (temp_int mod 8) IS
                    WHEN 0 => temp_oct(j) := 0;
                    WHEN 1 => temp_oct(j) := 1;
                    WHEN 2 => temp_oct(j) := 2;
                    WHEN 3 => temp_oct(j) := 3;
                    WHEN 4 => temp_oct(j) := 4;
                    WHEN 5 => temp_oct(j) := 5;
                    WHEN 6 => temp_oct(j) := 6;
                    WHEN 7 => temp_oct(j) := 7;
                END CASE;
            END LOOP;
        RETURN temp_oct;
    END FUNCTION;

    FUNCTION oct2int(oct : octal_number; octal_length : INTEGER) RETURN INTEGER IS
        VARIABLE temp_int : INTEGER := 0;
        BEGIN
            FOR j IN 0 TO octal_length LOOP
                temp_int := 8**j * temp_int + oct(j);
            END LOOP;
        RETURN temp_int;
    END FUNCTION;
END PACKAGE BODY octal_conversions;
```
Creating a Package

1. In the ModelAgent, select a library containing VHDL-AMS language resources. This type of library has a green package folder at the top of the model tree.
2. Choose ELEMENTS>NEW>PACKAGE. The package Properties dialog opens.
3. Click the «Package Text» tab and check the «Enable» box.
4. Enter a package name and click <OK>. A template with the specified package name appears in the dialog.

5. Enter the complete declaration of the package in VHDL-AMS syntax into the «Package Text» tab.
6. Click the «Package Body Text» tab and check the «Enable» box. A template with the specified package name appears in the dialog.
7. Enter the complete description of the package in VHDL-AMS syntax into the «Package Body Text» tab.
8. When all definitions are complete, click <OK>.
Inserting a Package from an Existing .vhd File

1. Open the ModelAgent in the SSC Commander.
2. Select a library for the new package.
3. Choose ELEMENTS>INSERT>TEXT MODEL(S) FROM VHDLA-FILE in the ModelAgent. The Open VHDLA Description dialog which is used to select a .vhd file opens.

   This command is available only if the VHDL-AMS resource is already installed in the library (check settings using FILE>PROPERTIES).
4. Select a .vhd file that includes a package definition. The Insert Model dialog displays all available package definitions in the file.

5. Select the package definitions in the tree.
6. Click <OK> to import the package.

Properties of Packages

The ELEMENT>PROPERTIES command opens the Properties dialog of a selected package. The package shortcut menu can also be used to open the dialog.

General Tab

The «General» tab contains the title of the package used in the model tree and the entry «Modified by». These settings have no function in other applications.

The name in «Modified by» is always replaced with the active user name when the ModelAgent is closed.

Package Text Tab

The «Package Text» tab displays the package declaration. The text can be directly edited here. Clicking <OK> checks the syntax of the text. Error messages are displayed below the input field in a message window.
Package Body Text Tab

The «Package Body Text» dialog displays the package body description. The text can be directly edited here. Clicking «OK» checks the syntax of the text. Error messages are displayed below the input field. Encoded package bodies have no text in this dialog.

Compile Function

The menu command File>Compile All Text Models compiles all VHDL-AMS models in the selected library. This feature is needed when model descriptions or packages have been changed. There is no message generated when the process is completed.
5.5 Simulator Environment

The VHDL-AMS simulator is one of several simulators that are integrated into the SIMPLORER system. It performs calculations on simulation models described in VHDL-AMS. The SML simulator starts the VHDL-AMS simulator if VHDL-AMS models are included in the Schematic sheet.

The VHDL-AMS solver has three components: an analog solver, a digital solver, and a controller. The analog solver is integrated with the SIMPLORER analog solver, while the digital solver is an independent event-drive VHDL engine. The controller determines the interaction between the analog and digital solver. For example, the controller initiates the digital solver if any events are present for digital signals. It controls the data flow and interaction between the analog and digital solvers and also proposes the simulation step size for the SIMPLORER Simulation Backplane. If there are pending events for the digital solver, the analog solver will evaluate also at those events. On the other hand, the digital solver is not initiated unless it has specific events to evaluate, even if the analog solver is evaluating.

VHDL-AMS models may be used in parallel with SIMPLORER models. All VHDL-AMS models are simulated by the analog solver and the digital solver. Consequently, the VHDL-AMS models do not display step delays along the simulator backplane, unlike the SML models. VHDL-AMS block models, as well as the circuit elements, are simulated by the analog solver, while SIMPLORER blocks are solved by a separate block diagram simulator. As a consequence, the simulation results from SML and VHDL-AMS models may not be identical. The difference in the results usually decreases if the minimum and maximum time step of the system, \( H_{\text{MIN}} \) and \( H_{\text{MAX}} \), are reduced. VHDL-AMS models in SIMPLORER also support AC or DC analysis.

Models for VHDL-AMS Simulation

- Models in the «AMS» tab
- Models in the «Digital» tab
- Some models in the «Tools» tab
- Models with implementation of VHDL-AMS description

A model can have both SML and VHDL-AMS descriptions. The modeling language to be used is specified in the model Properties dialog. Double-click the model and click the «Library» tab. Open the «Select modeling languages» drop-down menu and select either SML or VHDL-AMS. If VHDL-AMS is selected, an architecture can be selected from the «Select Architecture/Modeling Level» drop-down menu.
Parameters of VHDL-AMS Simulator

In Schematic, choose SIMULATION > PARAMETERS, and click the «TR» tab to set the parameters for the VHDL-AMS simulator.

The simulation parameters specified in the SIMULATION > PARAMETERS dialog are applicable for simulation of schematics with VHDL-AMS models also. The minimum step size (HMIN) specified should be smaller than the smallest digital delay of all models in any schematic. The digital solver uses 1 fs (1 femto second or 1.0e-15 sec) as the minimum resolution time for the VHDL simulator.

<table>
<thead>
<tr>
<th>General Simulation Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>THETA (23)</td>
</tr>
<tr>
<td>TEND (40m)</td>
</tr>
<tr>
<td>HMIN (10µ)</td>
</tr>
<tr>
<td>HMAX (1m)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuit Simulator Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLVER</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>LDF (0.1…10)</td>
</tr>
<tr>
<td>ITERATMAX (5…30)</td>
</tr>
<tr>
<td>IEMAX 1µ…100µ</td>
</tr>
<tr>
<td>VEMAX 1m…1</td>
</tr>
</tbody>
</table>
6 VHDL-AMS Language Fundamentals

VHDL-AMS (Very high-speed integrated circuit Hardware Description Language – Analog and Mixed Signal) is an IEEE standardized language used for describing digital, analog, and mixed-signal systems.

The VHDL-AMS language fundamentals provided in this chapter provide a quick reference guide to find a specific statement or the syntax for a specific statement needed to write VHDL-AMS code.


VHDL-AMS is case insensitive; upper case letters are equivalent to lower case letters. Reserved words are in UPPER case and shown in BOLD in this document.

Identifiers are simple names starting with a letter and may have letters and digits. The underscore character is allowed but not as the first or last character of an identifier.

A comment starts with two consecutive hyphens, "--", and continues until the end of the line.

The following table shows the syntax used in the Tutorial.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY</td>
<td>VHDL-AMS keyword</td>
</tr>
<tr>
<td>{expression}</td>
<td>optional entry</td>
</tr>
<tr>
<td>[name</td>
<td>string]</td>
</tr>
<tr>
<td>identifier{,..}</td>
<td>repeated entries</td>
</tr>
<tr>
<td>=&gt; &lt;= :=</td>
<td>assignment operators</td>
</tr>
<tr>
<td></td>
<td>simultaneous statement</td>
</tr>
</tbody>
</table>

This chapter contains information on:

- Design units
- Subprograms
- Declarations
- Concurrent Statements
- Sequential Statements
- Simultaneous Statements
- Identifiers, Literals, and Expressions
- Predefined Data Types
- Predefined Attributes
- Reserved Words
6.1 Design Units

The VHDL-AMS language allows the definition of models for analog, digital, and mixed signal circuits and systems in a standardized language. Design units (also library units) are segments of VHDL-AMS code that can be compiled separately and stored in a library.

An entity normally consists of five basic elements, or design units: entities, architectures, packages, package bodies, and configurations. Entities and architectures are the only two design units that must exist in any VHDL-AMS design description. Packages and configurations are optional.

### Elements of a VHDL-AMS Model

#### Model Description

- **ENTITY** (interface description)
  
- **ARCHITECTURE** (body description)

#### Packages

- **PACKAGE Declaration**
  
- **PACKAGE BODY** (constants, functions, components, ...)

#### Configurations

- **CONFIGURATION** (instantiation and binding information)

---

#### 6.1.1 Entities and Architectures

Each VHDL-AMS design description consists of an **ENTITY** declaration and one or more architectures. The **ENTITY** declaration defines the inputs to and outputs from the model, and any **GENERIC** parameters used by the different implementations. Each **ARCHITECTURE** defines a different implementation or behavior of a given design unit.
Entity Declaration

An entity declaration defines input to a model and outputs supported by the model, and generic parameters used by the different implementations of the model.

```
ENTITY entity_name IS
  GENERIC (generic_list); -- optional generic list
  PORT (port_list); -- input/output signal ports
END ENTITY name;
```

generic_list Specifies static information to be communicated to a model from its environment for all architectures. These include timing information (setup, hold, delay times), ambient information (temperature), and other parameters.

```
PORT (port_list); -- input/output signal ports
```

Port Modes

An `IN` port can be read but not updated within the module. An `IN` port cannot appear on the left hand side of a signal assignment.

An `OUT` port can be updated but not read within the module. An `OUT` port cannot appear on the right hand side of a signal assignment.

An `INOUT` port is bidirectional and can be both read and updated, with multiple update sources possible.

Signal objects can use the mode `IN`, `OUT`, and `INOUT`; quantity objects can use `IN` and `OUT` whereas terminals have no direction mode.

Ports of type `BUFFER` and `LINKAGE` are transformed to `INOUT` types in SIMPLORER.

```
ENTITY spring_tr IS
  GENERIC (s0: DISPLACEMENT := 0.0); -- list of generic parameters
  PORT QUANTITY c: IN STIFFNESS := 100.0; -- list of quantity ports
       TERMINAL tr1, tr2 : TRANSLATIONAL_V; -- list of terminal ports
       SIGNAL ctrl: IN BIT); -- list of signal ports
END ENTITY spring_tr;
```

Architecture

An architecture defines one particular implementation of a design unit (model behavior), at some desired level of abstraction.

```
ARCHITECTURE architecture_name OF entity_name IS
  declarations
BEGIN
  concurrent/sequential/simultaneous statements
END ARCHITECTURE [architecture_name]
```

declarations Information used in the model description. Declarations include data types, constants, signals, files, components, attributes, subprograms, and others.

```
declarations
```

concurrent statements Digital statements that are executed asynchronously with respect to each other. They describe a design unit at one or more levels of modeling abstraction, including dataflow, structural, and/or behavior.

```
BEGIN
  concurrent/sequential/simultaneous statements
```

sequential statements Statements that are executed in the order in which they appear. They define algorithms for the execution of a subprogram or process.

```
BEGIN
  concurrent/sequential/simultaneous statements
```

simultaneous statements Statements that are executed at the same time with respect to each other. They describe analog Differential Algebraic Equations (DAE).

```
BEGIN
  concurrent/sequential/simultaneous statements
```

ARCHITECTURE behav OF spring_tr IS
QUANTITY s: DISPLACEMENT; -- branch quantity declaration
BEGIN
  BREAK s => s0; -- value assignment
  f := c*s; -- model equation
  v := s'DOT; -- model equation
END ARCHITECTURE behav;
6.1.2 Packages

A VHDL-AMS package contains subprograms, constant definitions, and/or type definitions that may be used in one or more design units. Each package comprises a declaration part and a package body. The declaration part represents the portion of the package that is visible outside of that package.

Package Declaration

Package declarations define the available types, constants, natures, subprograms, and attributes.

```
PACKAGE package_name IS
  ...constant/type/subprogram/nature/attribute declarations
END package_name;
```

<table>
<thead>
<tr>
<th>declarations</th>
<th>Information used in the model description. Declarations include data types, constants, signals, files, components, attributes, subprograms, and others.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PACKAGE omnicaster_package IS</td>
<td></td>
</tr>
<tr>
<td>FUNCTION b2bl (b: BIT) RETURN BOOLEAN;</td>
<td></td>
</tr>
<tr>
<td>FUNCTION b2i (b: BIT; zv : INTEGER; ov : INTEGER) RETURN INTEGER;</td>
<td></td>
</tr>
<tr>
<td>END omnicaster_package;</td>
<td></td>
</tr>
</tbody>
</table>

| PACKAGE misc IS |
| TYPE short_integer IS range -100 TO 100; -- type declaration |
| CONSTANT K : REAL := 1.3806503e-23; -- constant declaration |
| SUBTYPE TEMPERATURE IS REAL TOLERANCE "DEFAULT_TEMPERATURE"; |
| SUBTYPE HEAT_FLOW IS REAL TOLERANCE "DEFAULT_HEAT_FLOW"; |
| NATURE THERMAL IS TEMPERATURE ACROSS HEAT_FLOW THROUGH THERM_REF REFERENCE; |
| END misc; |

Package Body

The package body defines the subprograms along with any internally-used constants and types.

```
PACKAGE BODY package_name IS
  ...subprogram bodies...
END package_name;
```

<table>
<thead>
<tr>
<th>subprogram_bodies</th>
<th>Specifies the subprograms declared in the package declaration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PACKAGE BODY omnicaster_package IS</td>
<td></td>
</tr>
<tr>
<td>FUNCTION b2bl (b : BIT) RETURN BOOLEAN IS</td>
<td></td>
</tr>
<tr>
<td>BEGIN</td>
<td></td>
</tr>
<tr>
<td>IF (b = '1') THEN</td>
<td></td>
</tr>
<tr>
<td>RETURN TRUE;</td>
<td></td>
</tr>
<tr>
<td>ELSE</td>
<td></td>
</tr>
<tr>
<td>RETURN FALSE;</td>
<td></td>
</tr>
<tr>
<td>END IF;</td>
<td></td>
</tr>
<tr>
<td>END b2bl;</td>
<td></td>
</tr>
<tr>
<td>FUNCTION b2i (b : BIT; zv : INTEGER; ov : INTEGER) RETURN INTEGER IS</td>
<td></td>
</tr>
<tr>
<td>BEGIN</td>
<td></td>
</tr>
<tr>
<td>IF (b = '1') THEN</td>
<td></td>
</tr>
<tr>
<td>RETURN ov;</td>
<td></td>
</tr>
<tr>
<td>ELSE</td>
<td></td>
</tr>
<tr>
<td>RETURN zv;</td>
<td></td>
</tr>
<tr>
<td>END IF;</td>
<td></td>
</tr>
<tr>
<td>END b2i;</td>
<td></td>
</tr>
<tr>
<td>END omnicaster_package;</td>
<td></td>
</tr>
</tbody>
</table>
Package Visibility

The **LIBRARY** statement is used to make specified libraries such as the **IEEE** library visible in a model description. If the library file name is not consistent with VHDL-AMS naming conventions, an alias needs to be defined. See “Alias for File Names” on page 209.

A **USE** statement can precede the declaration of any entity or architecture which is to utilize items from the package. If the **USE** statement precedes the **ENTITY** declaration, the package is also visible to the architecture.

To make all items of a package visible to a design unit, precede the desired design unit with a **USE** statement accompanied by the **ALL** keyword. To make single items of a package visible, only the corresponding item is defined in the **USE** statement. This saves simulation (compilation) time, since all visible items in a **USE** statement must be loaded before simulation.

See also “**WORK Library**” on page 208.

```vhdl
LIBRARY library_name;
USE library_name.package_name.ALL; -- all items are visible
USE library_name.package_name.item_name; -- one item is visible
```

**library_name** Library name, for example **transformations**. The extension .smd of SIMPLORER libraries must be omitted.

**package_name** Name of the package in a library, for example **omnicaster_package**.

**item_name** Name of a type, subprogram, or constant in the package.

```vhdl
LIBRARY TRANSFORMATIONS;  -- make library transformations.smd visible
LIBRARY VHDLAMS_TUTORIAL; -- make library vhdlams_tutorial.smd visible
USE TRANSFORMATIONS.omnicaster_package.ALL; -- all information available
USE VHDLAMS_TUTORIAL.octal_conversions.int2oct; -- function int2oct available
```

**VHDL-AMS Standard Packages and Types**

Several standard packages are installed along with SIMPLORER in the **STD** and **IEEE** libraries. These libraries can be found in the **AMS** tab in the ModelAgent. The VHDL-AMS source code for most of the packages can be viewed in the ModelAgent by opening the Properties dialog of each package. The source code for the **MATH_REAL**, **NUMERIC_STD** and **NUMERIC_BIT** packages are protected by an IEEE copyright and consequently cannot be viewed in the ModelAgent.

To use any of the packages, the **LIBRARY** statement and the **USE** statement need to be specified as follows:

```vhdl
LIBRARY library_name;
USE library_name.package_name.ALL;
```

**STD Library**

The **STD** library has two packages: **STANDARD** and **TEXTIO**.

The **STANDARD** package provides a number of types, subtypes, and functions. This package is different from all other packages in that it is always visible and need not be explicitly included within a model.

The **TEXTIO** package provides data types and subprograms that are required for reading and writing ASCII files.
The TEXTIO package can be made visible within a model description in the following manner:

```vhdl
LIBRARY STD;
USE STD.TEXTIO.ALL;
```

**The IEEE Library**

The IEEE library houses a number of packages that can be classified into three categories:

- Packages for the simulation of digital designs (Digital)
- Packages for the simulation of multidomain systems (Physical Domains)
- Packages for mathematical operations (Math)

**Packages for the Simulation of Digital Designs**

The following table lists the packages that are available for use in digital designs:

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Functionality</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD_LOGIC_1164</td>
<td>Defines the multi-value logic data type (STD_LOGIC) and associated operations.</td>
<td>USE IEEE.STD_LOGIC_1164.ALL;</td>
</tr>
<tr>
<td>STD_LOGIC_ARITH</td>
<td>Synopsys package based on multi-value logic that defines types and basic arithmetic operations for representing integers.</td>
<td>USE IEEE.STD_LOGIC_ARITH.ALL;</td>
</tr>
<tr>
<td>STD_LOGIC_SIGNED</td>
<td>Synopsys extension of the STD_LOGIC_ARITH package to handle multi-value logic values as signed integers.</td>
<td>USE IEEE.STD_LOGIC_SIGNED.ALL;</td>
</tr>
<tr>
<td>STD_LOGIC_UNSIGNED</td>
<td>Synopsys extension of the STD_LOGIC_ARITH library to handle multi-value logic values as unsigned integers.</td>
<td>USE IEEE.STD_LOGIC_UNSIGNED.ALL;</td>
</tr>
<tr>
<td>NUMERIC_STD</td>
<td>IEEE package based on multi-value logic that defines types and basic arithmetic operations for representing integers. This is similar to STD_LOGIC_ARITH and consequently should not be used together.</td>
<td>USE IEEE.NUMERIC_STD.ALL;</td>
</tr>
<tr>
<td>NUMERIC_BIT</td>
<td>IEEE package based on binary (BIT) data type that defines types and basic arithmetic operations for representing integers.</td>
<td>USE IEEE.NUMERIC_BIT.ALL;</td>
</tr>
</tbody>
</table>

The definitions for signed and unsigned data types are available in the NUMERIC_STD package, and in the Synopsys SIGNED and UNSIGNED packages. Consequently, the Synopsys packages cannot be used with the NUMERIC_STD package in any VHDL-AMS design.
### Packages for the Simulation of Multidomain Systems

The following table lists the physical domain packages that are available for use in multidomain simulations:

<table>
<thead>
<tr>
<th>Domain</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrical</td>
<td>USE IEEE.ELECTRICAL_SYSTEMS.ALL;</td>
</tr>
<tr>
<td>Mechanical</td>
<td>USE IEEE.MECHANICAL_SYSTEMS.ALL;</td>
</tr>
<tr>
<td>Thermal</td>
<td>USE IEEE.THERMAL_SYSTEMS.ALL;</td>
</tr>
<tr>
<td>Fluidic</td>
<td>USE IEEE.FLUIDIC_SYSTEMS.ALL;</td>
</tr>
<tr>
<td>Radiant</td>
<td>USE IEEE.RADIANT_SYSTEMS.ALL;</td>
</tr>
<tr>
<td>Energy Systems</td>
<td>USE IEEE.ENERGY_SYSTEMS.ALL;</td>
</tr>
<tr>
<td>Fundamental Constants</td>
<td>USE IEEE.FUNDAMENTAL_CONSTANTS.ALL;</td>
</tr>
<tr>
<td>Material Constants</td>
<td>USE IEEE.MATERIAL_CONSTANTS.ALL;</td>
</tr>
</tbody>
</table>

The following table lists the common declarations and constants that are valid in all physical domains:

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy Systems</td>
<td>USE IEEE.ENERGY_SYSTEMS.ALL;</td>
</tr>
<tr>
<td>Fundamental Constants</td>
<td>USE IEEE.FUNDAMENTAL_CONSTANTS.ALL;</td>
</tr>
<tr>
<td>Material Constants</td>
<td>USE IEEE.MATERIAL_CONSTANTS.ALL;</td>
</tr>
</tbody>
</table>

### Packages for Mathematical Operations

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math_Real (1076.2)</td>
<td>USE IEEE.MATH_REAL.ALL;</td>
</tr>
</tbody>
</table>
Subprograms define algorithms for calculating particular functions of a model. They can be used to divide complex model descriptions into smaller sections.

There are two forms of subprograms: procedures and functions. A procedure call is a statement; a function call is an expression and returns a value. A subprogram has two parts:

- Declaration statements
- Sequential statements defining the behavior

Subprograms can use constants, variables, and signals as parameters. The parameters that are used within a subprogram are called the formal parameters, while the parameters passed into the function or procedure are called the actual parameters.

Note that a simultaneous procedural statement is different from a subprogram. A procedural statement is used to describe analog behavior that occurs sequentially (rather than be solved simultaneously).

**Procedures**

A procedure defines a group of sequential statements that are executed when the procedure is called. A procedure can return any number of values (or no values) via its parameter list. A procedure call invokes the execution of the procedure body.

```
PROCEDURE procedure_name [(formal_parameters)] IS
  ...declarations
BEGIN
  ...sequential statements
END [PROCEDURE] [procedure_name];

...procedure name[actual_parameters] -- procedure call
```

**Formal parameters**

Specifies a list of parameters (constants, signals, or variables with the mode in, out, or inout).

**Declarations**

Declarations include data types, constants, signals, files, variables, attributes, and subprograms.

**Sequential statements**

Statements that are executed in the order in which they appear, that define algorithms for the execution of a subprogram or process.

```
-- declaration with formal parameters int and bin
PROCEDURE int2bin (VARIABLE int: IN INTEGER; VARIABLE bin: OUT BIT_VECTOR) IS
  VARIABLE temp: INTEGER;
BEGIN -- start of sequential procedure statements
  temp := int;
  FOR i IN 0 TO (bin’LENGTH -1) LOOP
    IF (temp mod 2 = 1) THEN
      bin(i) := '1';
    ELSE
      bin(i) := '0';
    END IF;
    temp := temp/2;
  END LOOP;
END int2bin; -- end of sequential procedure statements

PROCESS -- continued model description
  VARIABLE in_var: INTEGER:=10;
  VARIABLE out_vec: BIT_VECTOR (1 to 8);
BEGIN -- procedure call with actual parameters int_var and out_vec
  int2bin (in_var, out_vec);
  WAIT;
END PROCESS;
```
## Functions

A function defines a group of sequential statements that are executed when the function is called. A function returns a single value. Unlike procedures, functions are primarily used in expressions and only have inputs in their argument list. A function call invokes the execution of the function body.

```
FUNCTION function_name RETURN type_name
  -- Declarations
BEGIN
  -- sequential statements
  END FUNCTION function_name;
  function_name[actual_parameter] -- function call
END FUNCTION;
```

**PURE | IMPURE**

Pure functions return the same value each time they are called with the same values as actual parameters (default). Impure functions can return a different value each time they are called, even when multiple calls have the same actual parameter values.

**formal_parameters**

Specifies a list of parameters (constants, signals, or variables with the mode in, out, or inout).

**type_name**

Data type or data subtype name, for example BIT, INTEGER, REAL, … of the return value.

**declarations**

Declarations include data types, constants, signals, files, components, attributes, subprograms, and other information used in the model description.

**sequential statements**

Statements that are executed in the order in which they appear, that define algorithms for the execution of a subprogram or process.

```
-- declaration with formal parameters i and length and data type of return value
FUNCTION i2bv (i: INTEGER; length: INTEGER) RETURN BIT_VECTOR IS
  VARIABLE bv: BIT_VECTOR(length-1 DOWNTO 0);
  VARIABLE temp: INTEGER;
BEGIN
  temp := i;
  FOR j IN 0 TO (bv'LENGTH-1) LOOP
    IF (temp mod 2 = 1) THEN
      bv(j) := '1';
    ELSE
      bv(j) := '0';
    END IF;
    temp := temp/2;
  END LOOP;
  return bv;
END i2bv;
```

```
ARCHITECTURE bench OF entity_name IS
  SIGNAL in_var: INTEGER;
  SIGNAL out_vec: BIT_VECTOR (1 to 8);
BEGIN
  in_var <= 10;
  -- function call with actual parameters in_var and value 8
  out_vec <= i2bv (in_var,8) AFTER 1ms;
END ARCHITECTURE bench;
```
6.3 Declarations

Declarations specify types, data objects, attributes, and components that can be used in design units. Declarations can be located in packages, entities, and architectures. The scope or visibility of a declaration depends on where they are used.

6.3.1 TYPE Declarations

VHDL-AMS data types can be classified as scalar, composite, access and file types. VHDL-AMS supports different kinds of data objects for each data type. These data objects include constants, variables, signals, quantities, terminals, and files. VHDL-AMS includes a number of predefined data types, and allows user-defined data types as needed. A TYPE statement is used to declare a new type. A SUBTYPE statement is used to constrain an existing type.

Type Declarations

A type declaration defines a new data type.

- Scalar type declaration: Declares a type that can be used to create enumeration, integer, physical, and floating point elements.
- Composite type declaration: Declares a type for creating array or record elements.
- File type declaration: Declares a type for creating file handles.

| Type Name | Data Type or Data Subtype Name, for Example BIT, INTEGER, REAL, ...
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scalar_int1</td>
<td>IS RANGE -5 TO 5;</td>
</tr>
<tr>
<td>scalar_int2</td>
<td>IS RANGE 31 DOWNTO 0;</td>
</tr>
<tr>
<td>scalar_bit</td>
<td>IS ('0', '1');</td>
</tr>
<tr>
<td>scalar_enum</td>
<td>IS (red, green, blue);</td>
</tr>
<tr>
<td>scalar_boolean</td>
<td>IS (TRUE, FALSE);</td>
</tr>
<tr>
<td>composite1</td>
<td>IS ARRAY (0 TO 31) OF BIT;</td>
</tr>
<tr>
<td>composite2</td>
<td>IS ARRAY (natural RANGE &lt;&gt;) OF INTEGER;</td>
</tr>
<tr>
<td>composite3</td>
<td>IS RECORD</td>
</tr>
<tr>
<td></td>
<td>RE: REAL; IM: REAL;</td>
</tr>
<tr>
<td></td>
<td>END RECORD composite3;</td>
</tr>
<tr>
<td>composite4</td>
<td>IS ARRAY (INTEGER RANGE &lt;&gt;), INTEGER RANGE &lt;&gt; OF REAL;</td>
</tr>
<tr>
<td>file_txt</td>
<td>IS FILE OF integer;</td>
</tr>
</tbody>
</table>
SUBTYPE Declaration

A subtype declaration defines a type that is derived from an existing type. **TYPE** creates a new type while **SUBTYPE** creates a type that is a constraint of an existing type. It is possible to assign values of objects belonging to subtypes to objects belonging to the base type.

<table>
<thead>
<tr>
<th>SUBTYPE name</th>
<th>type_name [constraint] [tolerance];</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint</td>
<td>Specifies a range constraint for a scalar type with a ... RANGE ... TO (DOWN-TO) ... statement.</td>
</tr>
</tbody>
</table>

| SUBTYPE electrical_units | STRING (1 TO 20); |
| variable var_names: electrical_units := "voltage, current"; |
| SUBTYPE small_int | INTEGER RANGE 0 TO 10; -- subtype with constraint definition |
| VARIABLE little : small_int := 4; |

NATURE Declaration

A nature declaration defines a nature and its across and through quantities that can be accessed through the model terminals. See also “Across and Through Quantities of Natures” on page 12.

| NATURE nature_name | is_across_type_name ACROSS through_type_name THROUGH reference_terminal REFERENCE; |
| nature_name | Name of a nature, for example ELECTRICAL, MECHANICAL, THERMAL, ... |
| across_type_name | Specifies the branch types of the nature, which are used to define the type of branch quantities. Only terminals can represent a nature. |
| through_type_name | reference_terminal | Specifies the ground (zero) of the across type of a nature. |

| NATURE ELECTRICAL | IS -- nature declaration of an electrical system |
| voltage ACROSS current THROUGH electrical_ref REFERENCE; |
| NATURE THERMAL | IS -- nature declaration of a thermal system |
| temperature ACROSS heat_flow THROUGH thermal_ref REFERENCE; |

6.3.2 Data Object Declarations

A data object holds a value of a specified type. Every data object belongs to one of six different classes: Constant, Variable, Signal, File, Quantity, and Terminal.

CONSTANT Declaration

A constant declaration assigns a value to an identifier of a given data type. The use of constants can improve the readability of VHDL-AMS code. The value of a **CONSTANT** object cannot be changed by any executable code after declaration.

| CONSTANT constant_name: type_name [expression]; |
| constant_name | List of constant names separated by a comma. |
| type_name | Data type or data subtype name, for example BIT, INTEGER, REAL, ... |
| expression | Expression that performs an arithmetic or logical computation by applying an operator to one or more operands (constant value). |
CONSTANT Pi: REAL := 3.14159;
CONSTANT Half_Pi: REAL := Pi/2.0;
CONSTANT cycle_time: TIME := 11 ns;
CONSTANT N, N5: INTEGER := 5;
CONSTANT ctrl: BIT := '1'; -- logic 1 constant
CONSTANT zero4: BIT_VECTOR(0 to 3) := "0000";

A signal declaration defines an identifier as a signal object. A **SIGNAL** object holds a list of values, including the previous value, current value, and a set of possible future values that appear on the signal. A signal object has “digital properties”; this means that any change in a signal causes an event which can start a process or value assignment specified in the model description.

A constant declaration defines an identifier as a constant object. A **CONSTANT** object holds a single value, which can be a literal or a computation. Constants cannot change and are used to represent values that do not change during the simulation.

**SIGNAL** name_list: type_name [expression];

- **name_list** List of signal names separated by a comma.
- **type_name** Data type or data subtype name, for example BIT, INTEGER, REAL, ...
- **expression** Expression that performs an arithmetic or logical computation by applying an operator to one or more operands (default value).

**SIGNAL** a_bit : BIT := '0';
a_bit <= b_bit XOR '1';
**SIGNAL** clock : BIT := '0';
clock <= NOT clock AFTER 1 ms;
**SIGNAL** bool_sig : BOOLEAN := FALSE;
bool_sig <= TRUE WHEN clock = '1' ELSE FALSE;
**SIGNAL** gate_delay : TIME := 10 ns;

A variable declaration defines an identifier as a variable object. A variable can be of any scalar or aggregate data type and is updated immediately when an assignment statement is executed. Variables can be declared only within subprograms, procedural statements, or process statements.

**VARIABLE** variable_name: type_name [expression];

- **variable_name** List of variable names separated by a comma.
- **type_name** Data type or data subtype name, for example BIT, INTEGER, REAL, ...
- **expression** Expression that performs an arithmetic or logical computation by applying an operator to one or more operands (default value).

**VARIABLE** num_events : INTEGER := 0;
BEGIN
  num_events := num_events + 1;
END PROCESS;

A file declaration defines an identifier as a file object.
QUANTITY Declaration

A quantity declaration defines one or more identifiers as quantity objects. A quantity object is specified by its type and a default value. Quantities can be declared in both entity and architecture declaration placeholders. Quantities can be classified as PORT, FREE and BRANCH quantities depending on where they are declared. PORT quantities appear in the entity declaration, BRANCH quantities appear in the architecture declaration and specify the across and through values for terminals of a particular nature, and FREE quantities are declared as analog values in the architecture declaration.

```vhdl
QUANTITY name_list : real_type_name [expression];
QUANTITY [across_aspect] [through_aspect] terminal_aspect;
```

name_list List of quantities separated by a comma.
real_type_name Real type or real subtype name, for example REAL, VOLTAGE, TORQUE...
across_aspect Identifier that serves as the across quantity for the specified terminals.
through_aspect Identifier that serves as the through quantity for the specified terminals.
terminal_aspect Positive and negative terminal names.

```vhdl
ARCHITECTURE admittance OF load IS
  QUANTITY v ACROSS i THROUGH p TO m;
  QUANTITY ctrl_ramp: REAL := 0.0;
  QUANTITY ctrl_qty: REAL := 0.0;
BEGIN
  -- ...
END ARCHITECTURE admittance;
```

TERMINAL Declaration

A terminal declaration declares a terminal, and also the reference quantity and contribution quantity of the terminal.

```vhdl
TERMINAL name_list: nature_name;
```

name_list List of terminals separated by a comma.
nature_name Name of the conserved physical domain.

```vhdl
TERMINAL terminal_el : ELECTRICAL; --Local Electrical Terminal
TERMINAL terminal_t1 : THERMAL; --Local Thermal Terminal
```
6.3.3 Other Declarations

**ATTRIBUTE Declaration**

An attribute declaration defines an attribute name and its type. An attribute specification assigns a value to the attribute. See also Chapter 6.9, “Predefined Attributes” on page 199.

```
ATTRIBUTE attribute_name: type_name; -- declaration
ATTRIBUTE attribute_name OF name: entity_class IS expression; -- specification
```

- **attribute_name**: Name of the attribute.
- **type_name**: Data type name, for example BIT, INTEGER, REAL, ...
- **name**: Identifier of the existing object.
- **entity_class**: Specifies the name of the entity class, for example TYPE, SIGNAL, FILE ...
- **expression**: Expressions to perform an arithmetic or logical computation by applying an operator to one or more operands.

```
ATTRIBUTE UNIT OF VOLTAGE: SUBTYPE IS "Volt"; -- attribute declaration
ATTRIBUTE SYMBOL OF RESISTANCE: SUBTYPE IS "Ohm";
```

**COMPONENT Declaration**

A component declaration defines a component’s interface and is typically placed in an architecture or package declaration. The component or instances of the component are related to a design entity in a library using a configuration.

```
COMPONENT component_name [IS]
  [local_generic_clause]
  [local_port_clause]
END COMPONENT [component_name];
```

- **component_name**: Name of the component.
- **local_generic_clause**: Specifies static information to be communicated to a model from its environment in the form: variable_name : variable_type := value;
- **local_port_clause**: Specifies dynamic information to be communicated to a model from its environment in the form: variable_name : port_mode variable_type;

```
ENTITY inv4 IS
  GENERIC (TP_LH, TP_HL: TIME);
  PORT (I_4 : IN BIT_VECTOR(3 DOWNTO 0);
    Y_4 : OUT BIT_VECTOR(3 DOWNTO 0));
END ENTITY inv4;

ARCHITECTURE inv4_struct OF inv4 IS
  COMPONENT inv
    GENERIC (TP_LH, TP_HL: TIME);
    PORT (I1 : IN BIT;
      Y1 : OUT BIT);
  END COMPONENT;
  SIGNAL INV_OUT : BIT_VECTOR(3 DOWNTO 0);
BEGIN
  -- instantiation of the inv component
  INV_1 : inv GENERIC MAP (TP_LH,TP_HL) PORT MAP (I_4(0),Y_4(0));
  INV_2 : inv GENERIC MAP (TP_LH,TP_HL) PORT MAP (I_4(1),Y_4(1));
  INV_3 : inv GENERIC MAP (TP_LH,TP_HL) PORT MAP (I_4(2),Y_4(2));
  INV_4 : inv GENERIC MAP (TP_LH,TP_HL) PORT MAP (I_4(3),Y_4(3));
END inv4_struct;
```
6.4 Concurrent Statements

Concurrent statements are included within ARCHITECTURE bodies and within BLOCK statements, representing concurrent digital behavior within the modeled design unit. These statements are executed in an asynchronous manner, with no defined order, modeling the overall behavior or structure of a design.

- Block Statement
- Process Statement
- Concurrent Procedure Call
- Concurrent Assertion Statement
- Concurrent Signal Assignment Statement
- Component Instantiation Statement
- Generate Statement

BLOCK Statement

A block statement groups related concurrent statements. The order of the concurrent statements does not matter, because all statements are always executed together.

If a guard expression appears after the reserved word BLOCK, a boolean variable GUARD is automatically defined and set to the boolean value of the guard expression. GUARD can then be tested within the block, to perform selected signal assignments or other statements only when the guard condition evaluates to TRUE.

```
[label_name:] BLOCK [(guard expression)]
...local declarations
BEGIN
...concurrent statements
END BLOCK [label_name];
```

- guard expression Defines the value of the signal. The type of the expression must be BOOLEAN.
- local declarations Declarations include data types, constants, signals, files, components, attributes, subprograms, and other information used in the model description.
- concurrent statements Digital statements that are executed asynchronously with respect to each other, that describe a design unit in one or more modeling styles such as dataflow, structural, and/or behavior styles.

```
-- D Latch: Transfer D input to Q output when Enable = '1'
B1:
BLOCK [Enable = '1']
BEGIN
  Q <= guarded D AFTER 5ns;
END BLOCK B1;
```
**PROCESS Statement**

A **PROCESS** statement contains sequential statements but is itself a concurrent statement within an architecture. An independent sequential process represents the behavior of some portion of a design. The body of a process is a list of sequential statements. See also “Processes” on page 205.

The sequential statements in the process are executed in order, commencing with the beginning of simulation. After the last statement of a process has been executed, the process is repeated from the first statement, and continues to repeat until suspended. Processes can be suspended with a **WAIT** statement. The **WAIT** statement is a versatile statement that allows a process to be suspended for a specific period of time, boolean condition to occur, or an event to occur on a signal. The optional sensitivity list is equivalent to providing a **WAIT ON** statement and causes the process to be suspended. A process cannot have both a sensitivity list as well as a **WAIT ON** statement. See also “WAIT Statement” on page 184.

```vhdl
PROCESS (sensitivity_list)  
BEGIN  
  ...local declarations  
  ...sequential statements  
END PROCESS [label_name];
```

**sensitivity_list** List of signal names separated by a comma. The list represents a set of signals to which the **WAIT** statement within the process is sensitive.

If no sensitivity list is defined, a **WAIT** statement within the process must be specified.

**local declarations** Declarations declare data types, constants, signals, files, components, attributes, subprograms, and other information used in the model description.

**sequential statements** Statements that are executed in the order in which they appear, that define algorithms for the execution of a subprogram or process.

```vhdl
clock <= NOT clock AFTER 10 us;  
PROCESS (clock) -- sensitivity list consists of clock  
VARIABLE num_events : INTEGER := 0;  
BEGIN  
  num_events := num_events + 1;  
END PROCESS;
```

**Concurrent Procedure Call Statement**

A concurrent procedure call, which is used in an **ARCHITECTURE** or a **BLOCK** statement, invokes an externally defined subprogram with parameters passed to it as necessary. See also “Procedures” on page 172.

```vhdl
CALLED PROCEDURE procedure_name [actual_parameter_list];  
```

**actual_parameters** Values of these parameters are transferred to the procedure for the formal parameters (the parameters specified in the procedure declaration).

-- procedure ReadMem procedure can be defined in a package and then used in many places
ReadMem (DataIn, DataOut, RW, Clk);
Concurrent ASSERT Statement

A concurrent `ASSERT` statement checks a condition (occurrence of an event) and provides a report if the condition is `FALSE`. A severity level can be specified to generate several types of messages.

```vhdl
ASSERT (condition) [REPORT string] [SEVERITY severity_level];
```

- **condition**: Specifies an expression which evaluates to a BOOLEAN value (TRUE or FALSE). If the condition expression is `FALSE` (indicating the assertion failed), the text specified in the string expression is displayed.
- **string**: Defines the text string which appears in the message. String literals are one-dimensional arrays of characters enclosed in double quotation marks (" ").
- **severity_level**: Defines how the message appears and the effect on simulation.
  - `NOTE`: message output in the SIMPLORER info window.
  - `WARNING`: message output in a dialog and a request for user input to break or continue simulation.
  - `ERROR/FAILURE`: message output in the SIMPLORER info window and termination of simulation.

**Examples**

```vhdl
ASSERT (TS > 0.0) REPORT "Sample Time must be specified" SEVERITY ERROR;
```

```vhdl
ASSERT (UL >= LL) REPORT "Upper Limit must be greater than Lower limit" SEVERITY WARNING;
```

```vhdl
ASSERT (arg > 100.0) REPORT "Invalid Function argument " & REAL'IMAGE(arg) & " at time " & TIME'IMAGE(now) & "," SEVERITY INFO;
```

Concurrent SIGNAL Assignment Statement

A concurrent signal assignment statement, one of the most common signal assignments, represents a process that assigns values to signals and specifies logical relationship between different signals. There is no significance to the order in which the assignments appear in the description. There are two forms: conditional signal assignment and selected signal assignment statements. See also "SIGNAL Assignment Statement" on page 185.

```vhdl
[label_name]:
target_signal <= var; -- signal assignment;
target_signal <= var1 WHEN condition ELSE var2; -- conditional signal assignment
target_signal <= var3 WHEN condition var3 WHEN OTHERS; -- selected signal assignment
target_signal <= var4 AFTER 5ns; -- signal assignment with delay
```

- **target_signal**: Name of the signal which obtains the value.
- **conditional_signal_assignment**: Special form of signal assignment that assigns values only if a sequence of related conditions are true.
- **selected_signal_assignment**: Special form of signal assignment that assigns values only if a sequence of related conditions are true but differs in that the input conditions specified have no implied priority.
- **condition**: Specifies an expression which evaluates to a BOOLEAN value (TRUE or FALSE).

**Examples**

```vhdl
ARCHITECTURE behav OF test IS
SIGNAL A: BIT :='1';
SIGNAL B: BIT :='0';
BEGIN
  A <= B;
  B <= '1'; -- value of B=1 value of A=1
END;
```
Component Instantiation Statement

The component instantiation statement instantiates (creates an instance of) predefined components within an architecture. Each such component is first defined in the declaration section of that architecture, and then instantiated one or more times in the body of the architecture. See also “WORK Library” on page 208.

```vhdl
label_name: ENTITY [library_name.]entity_name [(architecture_name)]
  [GENERIC MAP (generic_map_list)]
  [PORT MAP (port_map_list)];
```

generic_map_list Specifies a list of generics (separated by a comma) which associate values with the formal generics in the corresponding component declaration or entity interface.

port_map_list Specifies a list of ports (separated by a comma) which associate signals, quantities, and/or terminals with the formal ports in the corresponding component declaration or entity interface.

```
LIBRARY vhdlams_tutorial;
LIBRARY IEEE;
USE vhdlams_tutorial.res; -- model res of the tutorial_vhdlams library is declared
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
ENTITY bat_multi IS
  GENERIC (factor : REAL := 1.0;
            v_init : VOLTAGE := 12.0);
  PORT (TERMINAL p,m : ELECTRICAL;
        QUANTITY v_out : OUT VOLTAGE := 0.0);
END ENTITY bat_multi;
ARCHITECTURE struct OF bat_multi IS
  CONSTANT ri: RESISTANCE := 1.0e-2;
  CONSTANT fc: CAPACITANCE := 60.0;
  CONSTANT rd: RESISTANCE := 4.0e-2;
  CONSTANT sc: CAPACITANCE := 2.0e4;
  TERMINAL t1,t2 : ELECTRICAL;
  QUANTITY v ACROSS p TO m;
BEGIN
  fc1: ENTITY WORK.cap(behav) -- instantiation of a component in the WORK lib
      GENERIC MAP (c_nom => fc*factor, v_init => v_init)
      PORT MAP (p => t1, m => m);
  sc1: ENTITY WORK.cap(behav) -- instantiation of a component in the WORK lib
      GENERIC MAP (c_nom => sc*factor, v_init => v_init)
      PORT MAP (p => t2, m => m);
  ri1: ENTITY res(behav) -- instantiation of a component declared in the entity
      GENERIC MAP (r_nom => ri)
      PORT MAP (p => p, m => t1);
  rd1 : ENTITY res(behav) -- instantiation of a component declared in the entity
       GENERIC MAP (r_nom => rd)
      PORT MAP (p => t1, m => t2);
      v_out => v;
END ARCHITECTURE struct;
```
BREAK Statement

A **BREAK** statement indicates explicitly the occurrence of discontinuities in a VHDL-AMS description. The concurrent **BREAK** statement represents a process containing a **BREAK** statement.

```vhdl
ARCHITECTURE behav OF LMT IS
  SIGNAL lower_crossing : BOOLEAN := FALSE;
  SIGNAL upper_crossing : BOOLEAN := FALSE;
BEGIN
  BREAK ON lower_crossing;
  BREAK ON upper_crossing;
END ARCHITECTURE;
```

**BREAK** [label_name:] [break_list] WHEN condition;

<table>
<thead>
<tr>
<th><strong>break_list</strong></th>
<th>Specifies a list of break elements (separated by a comma) which can cause a discontinuity.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sensitivity_clause</strong></td>
<td>Specifies a list of signals to which the <strong>BREAK</strong> statement is sensitive.</td>
</tr>
<tr>
<td><strong>condition</strong></td>
<td>Specifies an expression which evaluates to a BOOLEAN value (<strong>TRUE</strong> or <strong>FALSE</strong>).</td>
</tr>
</tbody>
</table>
6.5 Sequential Statements

Sequential statements appear in process statements and in subprograms (procedures and functions), representing sequential behavior within the modeled design unit. These statements are executed in the order in which they appear in the model.

- WAIT Statement
- ASSERT Statement
- SIGNAL Assignment Statement
- VARIABLE Assignment Statement
- Procedure Call Statement
- IF Statement
- CASE Statement
- LOOP Statement
- NEXT Statement
- EXIT Statement
- RETURN Statement
- NULL Statement
- BREAK Statement

**WAIT Statement**

A **WAIT** statement suspends subprogram execution until a signal changes, a condition becomes *TRUE*, or a defined time period has elapsed. Combinations of these can also be used.

**WAIT** [ON sensitivity_list] [UNTIL condition] [FOR time_expression];

- **sensitivity_list** List of signals which can cause an event.
- **condition** Specifies an expression which evaluates to a BOOLEAN value (*TRUE* or *FALSE*).
- **time_expression** Specifies a time expression which evaluates to a TIME value.

```vhdl
WAIT ON INPUT;
WAIT UNTIL ctrl > 1.5;
WAIT FOR TDELAY*unit_time;
```
ASSERT Statement

A sequential **ASSERT** statement checks a condition and provides a report if the condition is not **TRUE**. A severity level can be specified to generate several types of messages.

```
ASSERT (condition) [REPORT string] [SEVERITY severity_level];
```

- **condition** Specifies an expression which evaluates to a BOOLEAN value (**TRUE** or **FALSE**). If the condition expression is **FALSE** (indicating the assertion failed), the text specified in the string expression is displayed.
- **string** Defines the text string which appears in the message. String literals are one-dimensional arrays of characters enclosed in double quotation marks (“ ”).
- **severity_level** Defines how the message appears and the effect for simulation.
  - **NOTE**: message output in the SIMPLORER info window.
  - **WARNING**: message output in a dialog and a request for user input to break or continue simulation.
  - **ERROR/FAILURE**: message output in the SIMPLORER info window and termination of simulation.

```
ENTITY sequential_assert IS
END;
ARCHITECTURE behav OF sequential_assert IS
SIGNAL clk:bit;
BEGIN
  clk<=not clk AFTER 1 ns;
  PROCESS(clk)
    variable a:integer := 0;
    BEGIN
      a:=a+1;
      ASSERT a/=1 REPORT "a!=1" SEVERITY FAILURE;
      ASSERT a/=2 REPORT "a!=2" SEVERITY WARNING;
      ASSERT a/=3 REPORT "a!=3" SEVERITY ERROR;
      ASSERT a<10 REPORT "a==" & INTEGER'IMAGE(a) SEVERITY NOTE;
    END PROCESS;
END;
```

SIGNAL Assignment Statement

A signal assignment statement assigns a waveform to one signal driver (edits the event queue). Signal assignments are always performed at the end of a process. See also “Concurrent SIGNAL Assignment Statement” on page 181 and “Signal Assignments with Delay” on page 205.

```
[label_name:]
target_signal <= [delay_mechanism] source_signal
```

- **target_signal** Name of the signal which obtains the value.
- **delay_mechanism** Specifies a delay for signal assignment after one of the reserved words: **TRANSPORT**, **REJECT**, or **INERTIAL**.
- **source_signal** Name of the signal which provides the value.

```
SIGNAL A: BIT :='1';
SIGNAL B: BIT :='0';
BEGIN
  A <= B;
  B <= '1'; -- value of B=1 value of A=0
  WAIT;
END PROCESS;
```
### VARIABLE Assignment Statement

A variable assignment updates a process, procedure, or function variable with the value of an expression. The update takes effect immediately, unlike signal assignments where there is at least a delta delay for the update to be reflected.

| variable_name | Name of the variable. |
| expression | Expression that performs an arithmetic or logical computation by applying an operator to one or more operands. |

```vhdl
clock <= NOT clock AFTER 10 us;
PROCESS (clock)
  VARIABLE num_events : INTEGER := 0;
BEGIN
  num_events := num_events + 1;
END PROCESS;
```

### Procedure Call Statement

A procedure call statement invokes an externally-defined subprogram in the same manner as a concurrent procedure call. A sequential procedure call statement differs from a concurrent procedure call in that it is placed in a process, procedure, or function, and is executed in the order in which it appears. See also Chapter 6.2, “Subprograms” on page 172.

| procedure_name | Specifies a list of parameters (constants, signals, or variables with the mode in, out, or inout). |

```vhdl
-- procedure call with actual parameters int_var and out_vec
int2bin (in_var, out_vec);
```

### IF Statement

An **IF . THEN . ELSE** statement performs a sequence of statements depending on the defined condition. **ELSIF** and **ELSE** clauses are optional.

| condition | Specifies an expression wh ich evaluates to a BOOLEAN value (*TRUE* or *FALSE*). If the condition expression is *TRUE*, the corresponding statements after the condition are executed. |
| sequential statements | Statements that are executed in the order in which they appear, that define algorithms for the execution of a subprogram or process. |

```vhdl
pwm_ctrl:
PROCESS (cond1,cond2)
BEGIN
  IF (cond1) THEN
    ctrl_sig <= 0.0;
    ELSIF (cond2 AND (NOT cond1)) THEN
      ctrl_sig <= 1.0;
  END IF;
END PROCESS pwm_ctrl;
```
CASE Statement

A **CASE** statement selects one of a number of alternative sequences of statements for execution based on the value of an expression. The choices must be constants of the same discrete type as the expression. Case choices can be expressions or ranges. Case statements must also include all possible values of the control expression. The **OTHERS** expression can be used to guarantee that all conditions are covered.

```vhdl
CASE control_expression IS
  WHEN choice1 =>
    ...sequential statements
  WHEN choice2 =>
    ...sequential statements
  WHEN OTHERS =>
    ...sequential statements
END CASE [LABEL];
```

- **control_expression** Specifies a value that selects one statement sequence among the list of alternatives. The expression must be of a discrete type, or of a one-dimensional array.
- **choice** A choice specifies the value of the control expression for which the alternative is chosen. Each choice in a case statement alternative must be of the same type as the expression.
- **sequential statements** Statements that are executed in the order in which they appear, that define algorithms for the execution of a subprogram or process.

```vhdl
CASE J_K IS
  WHEN "11" =>
    state <= NOT state;
  WHEN "10" =>
    state <= '1';
  WHEN "01" =>
    state <= '0';
  WHEN OTHERS =>
    NULL;
END CASE;
```

LOOP Statements

The **WHILE** and **FOR** loop statements control the repetition of sequentially executed statements. Loop termination statements allow termination of one iteration, loop, or procedure.

```vhdl
NEXT WHEN condition]; -- end current loop iteration
EXIT WHEN condition]; -- exit innermost loop entirely
```

```vhdl
LOOP
  ...sequential statements -- use exit statement to leave the loop
END LOOP [label];
```

```vhdl
FOR variable IN start_val TO end_val LOOP
  ...sequential statements
END LOOP [LABEL];
```

```vhdl
WHILE condition LOOP
  ...sequential statements
END LOOP [LABEL];
```

- **sequential statements** Statements that are executed in the order in which they appear, that define algorithms for the execution of a subprogram or process.
- **variable/start_val/ end_val** Implicit index variables of the loop. Index variables need not be declared separately.
### NEXT Statement

A **NEXT** statement causes the next iteration in a loop.

```vhdl-ams
NEXT [label_name1:] [WHEN condition];
```

<table>
<thead>
<tr>
<th>condition</th>
<th>Specifies an expression which evaluates to a BOOLEAN value (TRUE or FALSE). If the condition is TRUE, the next iteration in the loop is executed.</th>
</tr>
</thead>
</table>

- NEXT;
- NEXT outer_loop;
- NEXT WHEN A>B;
- NEXT this_loop WHEN C=D OR done; -- done is a BOOLEAN variable

### EXIT Statement

An **EXIT** statement causes the immediate termination of the loop.

```vhdl-ams
EXIT [label_name] [WHEN condition];
```

<table>
<thead>
<tr>
<th>condition</th>
<th>Specifies an expression which evaluates to a BOOLEAN value (TRUE or FALSE). If the condition is TRUE, the loop is terminated.</th>
</tr>
</thead>
</table>

- EXIT;
- EXIT outer_loop;
- EXIT WHEN A>B;
- EXIT this_loop WHEN C=D OR done; -- done is a BOOLEAN variable
RETURN Statement

A RETURN statement terminates a subprogram. A function definition requires a return statement to specify the return value. In a procedure definition, a return statement is optional.

```
[label_name:] RETURN [expressions];
```

expression | Expression that performs an arithmetic or logical computation by applying an operator to one or more operands.

```
RETURN; -- from somewhere in a procedure
RETURN a+b; -- returned value in a function
```

NULL Statement

A NULL statement explicitly states that no action is required. It is often used in CASE statements because all choices must be covered, even if some of the choices are ignored.

```
[label_name:] NULL;
```

```
CASE J_K IS
  WHEN "11" =>
    state <= NOT state;
  WHEN "10" =>
    state <= '1';
  WHEN "01" =>
    state <= '0';
  WHEN OTHERS =>
    NULL;
END CASE;
```

BREAK Statement

A BREAK statement indicates explicitly the occurrence of discontinuities in a VHDL-AMS description. The sequential BREAK statement represents a process containing a BREAK statement.

The execution of a BREAK statement notifies the analog solver that it must determine the discontinuity augmentation set for the next analog solution point. It can also specify reset and initial values. The effect is conditional if the statement includes a condition.

```
[label_name:] BREAK [break_list] [WHEN condition];
```

break_list | Specifies a list of break elements (separated by a comma) which can cause a discontinuity.

condition | Specifies an expression which evaluates to a BOOLEAN value (TRUE or FALSE).
LIBRARY IEEE;
USE IEEE.MATH_REAL.ALL;

ENTITY sequential_break1 IS
END ENTITY sequential_break1;

ARCHITECTURE behav OF sequential_break1 IS
  SIGNAL xs1,xs2 : BOOLEAN := FALSE;
  SIGNAL aVal : REAL := 0.0;
BEGIN
  xs1 <= NOT xs1 AFTER 1 ms;
  xs2 <= NOT xs2 AFTER 2.5 ms;
  PROCESS (xs1, xs2)
  BEGIN
    BREAK aVal => 4.0 WHEN xs2;
    BREAK aVal => 2.0 WHEN xs1;
  END PROCESS;
END ARCHITECTURE behav;
6.6 Simultaneous Statements

Simultaneous statements appear in the architecture body of a model and can be placed in the same parts of a VHDL-AMS description as concurrent statements. Simultaneous statements are used to express Differential Algebraic Equations (DAE) that together with implicit equations describe the analog behavior of a model. The five types of simultaneous statements available are:

- Simple Simultaneous Statement
- Simultaneous IF Statement
- Simultaneous CASE Statement
- Simultaneous PROCEDURAL Statement
- Simultaneous NULL Statement

Simple Simultaneous Statement

A simple simultaneous statement specifies expressions that constrain the values of quantities by the analog solver.

```
expression == expression;
```

**expressions** Expressions to perform an arithmetic or logical computation by applying an operator to one or more operands.

```
TERMINAL plus, minus : ELECTRICAL;
QUANTITY voltage ACROSS current THROUGH plus TO minus;
QUANTITY power : REAL;
power == voltage * current; -- power calculation
```

Simultaneous IF Statement

A simultaneous IF statement specifies analog behavior of a system based on a set of conditions. A simultaneous IF statement differs from a sequential IF statement in that the simultaneous statement syntax is "IF USE- END USE" while the sequential statement syntax is "IF THEN END IF."

```
IF condition USE
  ...simultaneous_statements
ELSIF condition USE
  ...simultaneous_statements
ELSE
  ...simultaneous_statements
END USE [label_name];
```

**condition** Specifies an expression which evaluates to a BOOLEAN value (TRUE or FALSE). If the condition expression is TRUE, the corresponding statements after the condition are executed.

**simultaneous_statements** Statements that are executed at the same time with respect to each other, that describe Analog Differential Algebraic Equations (DAE).

```
IF (sw_on) AND (CTRL > 0.0) USE
  v == 0.0;
ELSE
  i == 0.0;
END USE;
```
**Simultaneous CASE Statement**

A simultaneous `CASE` statement specifies analog behavior by selecting one of a number of alternatives based on the value of an expression. A simultaneous `CASE` statement differs from a sequential `CASE` statement in that the simultaneous statement syntax is "CASE-USE-END CASE" while the sequential statement syntax is "CASE-IS-END CASE."

```vhdl
[variable_name:] CASE control_expression USE WHEN choices => ...simultaneous_statements { WHEN choices => ...simultaneous_statements} END CASE [LABEL];
```

- **control_expression** Specifies a value that selects one statement sequence among the list of alternatives. The expression must be of a discrete type, or of a one-dimensional array.
- **choice** A choice specifies the value of the control expression for which the alternative is chosen. Each choice in a case statement alternative must be of the same type as the expression.
- **simultaneous statements** Statements that are executed at the same time with respect to each other, that describe Analog Differential Algebraic Equations (DAE).

```vhdl
LIBRARY IEEE;
USE IEEE.MATH_REAL.ALL;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
ENTITY simultaneous_case IS END ENTITY simultaneous_case;
ARCHITECTURE behav OF simultaneous_case IS
TERMINAL n1, n2 : ELECTRICAL;
QUANTITY vin ACROSS iin THROUGH n1 TO electrical_ref;
QUANTITY vout ACROSS iout THROUGH n2 TO electrical_ref;
CONSTANT Amp : REAL := 1.0;
CONSTANT a : REAL := 1.0E3;
CONSTANT f : REAL := 1.0E3;
SIGNAL clk : INTEGER := 0;
BEGIN
clk <= clk+1 AFTER 1ms;
vout == vin'SLEW(1.0e38,-1.0e38);
CASE clk USE WHEN 0 => vin  == Amp*1.0*sin(2.0*math_pi*f*15.0*NOW);
WHEN 1 => vin  == Amp*2.0*sin(2.0*math_pi*f*15.0*NOW);
WHEN 2 => vin  == Amp*3.0*sin(2.0*math_pi*f*15.0*NOW);
WHEN 3 => vin  == Amp*4.0*sin(2.0*math_pi*f*15.0*NOW);
WHEN 4 => vin  == Amp*5.0*sin(2.0*math_pi*f*15.0*NOW);
WHEN OTHERS => vin  == Amp*6.0*sin(2.0*math_pi*f*15.0*NOW);
END CASE;
END ARCHITECTURE simple;
```
Simultaneous PROCEDURAL Statement

A simultaneous procedural statement provides a sequential notation for expressing Differential and Algebraic Equations. Procedural statements are included within the architecture of a model and are not invoked with a calling mechanism.

```vhdl
ENTITY fktarccos IS
    GENERIC (TS : REAL := 0.0);
    PORT (QUANTITY INPUT : IN REAL;
          QUANTITY VAL : OUT REAL);
END ENTITY fktarccos;

ARCHITECTURE behav OF fktarccos IS
    QUANTITY temp_val : REAL := 0.0;
BEGIN
    PROCEDURAL BEGIN
        IF (INPUT>-1.0) AND (INPUT<1.0) THEN
            temp_val := arccos(INPUT);
        ELSIF (INPUT = -1.0) THEN
            temp_val := MATH_PI;
        ELSE
            temp_val := 0.0;
        END IF;
    END PROCEDURAL;
    VAL := temp_val'ZOH(TS);
END behav;
```
A simultaneous `NULL` statement specifies explicitly that no action needs to be performed.

```vhdl
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
ENTITY two_port IS
  GENERIC
    (param : REAL := 1.0;
      mod_type : INTEGER := 0);
    -- RESISTOR(0); CAPACITOR(1);
    -- INDUCTOR(2)
  PORT
    (TERMINAL p,m : ELECTRICAL);
END ENTITY two_port;
ARCHITECTURE behav OF two_port IS
  QUANTITY v ACROSS i THROUGH p TO m;
BEGIN
  CASE mod_type USE
    WHEN 0 => v == i*param;
    WHEN 1 => i == param*v'dot;
    WHEN 2 => v == param*i'dot;
    WHEN OTHERS => NULL;
  END CASE;
END ARCHITECTURE behav;
```
6.7 Identifiers, Literals, and Expressions

Identifiers and literals are operands representing values of constants, variables, functions and so on in expressions. Expressions perform arithmetic or logic computations by applying an operator to one or more operands.

Identifiers

An identifier is the name of a constant, variable, function, signal, entity, port, subprogram, or parameter and returns that object’s value to an operand.

Identifiers in VHDL-AMS must begin with a letter, and can comprise any combination of letters, digits, and underscores. Identifiers must not end with an underscore and must not include two successive underscore characters. Also no space is allowed within an identifier since a space is a separator.

An indexed identifier is the name of one element of an array variable or signal. The expression must return a value within the array’s index range. The value returned to an operator is the specified array element.

```
letter { [underscore] letter_or_digit }  -- identifier
letter { [underscore] letter_or_digit } (expressions)  -- indexed identifier
```

```
Voltage1, power_dissipated, product_of_sums
voltage(2), current(3+1)
```

Literals

A literal (constant) operand can be a numeric, a character, an enumeration, or a string literal.

There are two forms of numeric literals: integer and real literals. Integer literals represent a whole number. Real literals can represent fractional numbers and always include a decimal point preceded and followed by at least one digit.

Both types of numeric literals can use exponential notation and can be expressed in a base. A decimal literal is written in base 10 and a based literal is written in a base from 2 to 16 and is composed of the base number, an octothorpe (#), the value in the given base, and another octothorpe (#).

```
base#digits# -- base must be a decimal number
"#101# -- decimal 5
16#A#16#E+2 -- floating, exponent is decimal
```

Character literals are single characters enclosed in single quotation marks, for example ‘a’.

```
Character literals are used both as values for operators and in defining enumerated types, such as CHARACTER and BIT.
```

Enumeration literals are values of enumerated types. The two kinds of enumeration literals are character literals and identifiers. Character literals are described earlier. Enumeration identifiers are those listed in an enumeration type definition. If two enumerated types use the same literals, those literals are overloaded.
### Arithmetic and Logical Expressions

Expressions in VHDL-AMS are similar to those of most high-level languages. Data elements must be of the same type, or subtypes of the same base type. Expressions perform arithmetic or logical computations by applying an operator to one or more operands. Operators specify the computation to perform, operands are the data for the computation.

#### Logical
- Predefined operators for types BIT and BOOLEAN.
  - AND | OR | NAND | NOR | XOR | XNOR | NOT

#### Relational
- Predefined operators for all types except files.
  - = | /=
- Predefined operators for scalar and discrete array types.
  - < | <= | > | >=

Relational operators include tests for equality (=), inequality (/=), and ordering of operands (<, <=, >, >=). The operands of each relational operator must be of the same type. The result type of each relational operator is the predefined type BOOLEAN.

#### Shift
- all | srl Shift left/right logical.
- sla | sia Shift left/right arithmetic.
- rol | ror Rotate left/right logical.

Operators for any one-dimensional array type whose element type is either of the predefined types BIT or BOOLEAN (left operand) and predefined type INTEGER (right operand). The result type of each shift operator is the same as the left operand.
### Adding
- Predefined operators for any numeric type.
  - `+`, `-`
- Predefined operator for any one-dimensional array type.
  - `&`
    - Concatenate, `a & b` makes one array

For each of these adding operators, the operands and the result are of the same type.

### Multiplying
- Predefined for any integer and any floating point type.
  - `*`, `/`
- Predefined for any integer type.
  - `mod`, `rem`
    - `a mod b` takes sign of `b`, `a rem b` takes sign of `a`

For each of these multiplying operators, the operands and the result are of the same type.

### Miscellaneous
- Predefined for any numeric type.
  - `abs`
    - Absolute value
  - `**`
    - `a**` is `a²`

The exponential operator `**` is predefined for each integer type and for each floating point type.
6.8 Predefined Data Types

VHDL-AMS includes a number of predefined data types.

Predefined VHDL-AMS Standard Data Types

The following table shows predefined types from the package STANDARD. The data type names are not technically reserved words but they represent a common standard. To avoid misunderstandings, do not re-define them. The standard package file is provided on the tutorial CD.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>'1', '0'</td>
<td>Q &lt;= '1';</td>
</tr>
<tr>
<td>BIT_VECTOR</td>
<td>Array of bits.</td>
<td>DataOut &lt;= &quot;0010101&quot;;</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>TRUE, FALSE</td>
<td>EQ &lt;= True;</td>
</tr>
<tr>
<td>INTEGER</td>
<td>-2, -1, 0, 1, 2, 3, 4 ...</td>
<td>Count &lt;= Count + 2;</td>
</tr>
<tr>
<td>REAL</td>
<td>1.0, -1.0E5</td>
<td>V1 = V2 / 5.3</td>
</tr>
<tr>
<td>TIME</td>
<td>1 ns, 7 ns, 100 ps</td>
<td>Q &lt;= '1' after 6 ns;</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>'a', 'b', '2', '$' ...</td>
<td>CharData &lt;= 'X';</td>
</tr>
<tr>
<td>STRING</td>
<td>Array of characters.</td>
<td>Msg &lt;= &quot;MEM: &quot; &amp; Addr</td>
</tr>
</tbody>
</table>

Predefined Type Declarations

```vhdl
TYPE INTEGER IS RANGE -2147483647 TO -2147483648;
TYPE BOOLEAN IS (FALSE, TRUE);
TYPE BIT IS ('0', '1');
TYPE CHARACTER IS (character_set_used_by_the_system);
TYPE SEVERITY_LEVEL IS (note, Warning, error, failure);
TYPE REAL IS RANGE -9.9e99 TO 9.9e99;
TYPE TIME IS RANGE -9.9e99 TO 9.9e99
  UNITS fs;
  ps = 1000 fs;
  ns = 1000 ps;
  us = 1000 ns;
  ms = 1000 us;
  sec = 1000 ms;
  min = 60 sec;
  hr = 60 min;
END UNITS;

TYPE DOMAIN_TYPE IS (QUIESCENT_DOMAIN, TIME_DOMAIN, FREQUENCY_DOMAIN);
TYPE STRING IS ARRAY (1 TO 2147483647) OF CHARACTER;
TYPE BIT_VECTOR IS ARRAY (0 TO 2147483647) OF BIT;
```
### 6.9 Predefined Attributes

An object attribute returns information about a signal or data type. Predefined attributes denote values, functions, types, and ranges associated with various kinds of named entities.

The syntax of an attribute is some named entity followed by an apostrophe and one of the following attribute names. A parameter list is used with some attributes. The following abbreviations are used in the tables:

- **Q** represents a quantity
- **S** represents a signal
- **X** represents a signal or variable
- **T** represents a type
- **A** represents an array or constrained array type
- **t** represents an expression for time
- **e** represents a static expression

#### Quantity Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q'DOT</td>
<td>value of the derivative with respect to time of Q at the time the attribute is evaluated</td>
</tr>
<tr>
<td>Q'INTEG</td>
<td>value of the time integral of Q from time 0 to the time the attribute is evaluated</td>
</tr>
<tr>
<td>Q'DELAYED[(t)]</td>
<td>value of quantity Q delayed by t; if t is omitted, it defaults to 0.0</td>
</tr>
<tr>
<td>Q'ABOVE(e)</td>
<td>true if Q-e is sufficiently larger than 0.0, false if Q-e is sufficiently smaller than 0.0. This attribute always returns a boolean signal (TRUE/FALSE).</td>
</tr>
<tr>
<td>Q'ZOH(e[,INITIAL_DELAY])</td>
<td>constant value of Q at the sample times INITIAL_DELAY+k*e until the next sample</td>
</tr>
<tr>
<td>Q'LTF(NUM,DEN)</td>
<td>Laplace transfer function of Q with NUM as the numerator and NEN as denominator</td>
</tr>
<tr>
<td>Q'ZTF(NUM,DEN,e[,INITIAL_DELAY])</td>
<td>Z transfer function of Q with NUM as the numerator and DEN as denominator</td>
</tr>
<tr>
<td>Q'SLEW[(MAX_RISING_SLOPE[,MAX_FALLING_SLOPE])]</td>
<td>quantity which follows the corresponding value of Q, but its derivative time is limited by MAX_RISING_SLOPE and MAX_FALLING_SLOPE</td>
</tr>
</tbody>
</table>

#### Signal Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S'RAMP[(TRISE[,TFALL])]</td>
<td>quantity which follows the corresponding value of S with delay of rise time and fall time; if TRISE or TFALL is greater than 0.0, the corresponding value change is linear from the current value of S to its new value, whenever S has an event</td>
</tr>
</tbody>
</table>

### S'CLEM(REAL'LOW,S'CL) | quantity which follows the corresponding value of S with rising S'CLEM(REAL'LOW,S'CL) slope and falling slope; if RISING_SLOPE is less than REAL'LOW, or if the value of FALLING_SLOPE is greater than REAL'LOW, the corresponding value change is linear from the current value of S to its new value, whenever S has an event |

### Q'SLEW(MAX_RISING_SLOPE[,MAX_FALLING_SLOPE]) | quantity which follows the corresponding value of Q, but its derivative time is limited by MAX_RISING_SLOPE and MAX_FALLING_SLOPE |
**VHDL-AMS Language Fundamentals**

### Signal Attributes

- `S'DELAYED[(t)]` -- value of signal S at time now-t; if t is omitted, it defaults to 0.0
- `S'STABLE` -- true if no event is occurring on signal S
- `S'QUIET` -- true if signal S is quiet (no event this simulation cycle)
- `S'QUIET(t)` -- true if signal S has been quiet for t time units
- `S'TRANSACTION` -- bit value which toggles each time when signal S changes
- `S'SEVENT` -- true if an event has occurred on signal S in the current cycle
- `S'ACTIVE` -- true if signal S is active in the current cycle
- `S'LAST_EVENT` -- time since the last event on signal S
- `S'LAST_ACTIVE` -- time since signal S was last active
- `S'QUIET(t)` -- true if signal S has been quiet for t time units
- `S'DRIVING` -- false if the current driver of signal S is a null transaction
- `S'DRIVING_VALUE` -- current driving value of signal S

### Data Type Bounds

- `T'BASE` -- base type of data type T
- `T'LEFT` -- left bound of data type T (largest if downto)
- `T'RIGHT` -- right bound of data type T (smallest if downto)
- `T'HIGH` -- upper bound of data type T (may differ from left bound)
- `T'LOW` -- lower bound of data type T
- `T'ASCENDING` -- true if range of T defined with to

### Enumeration Data Types

- `T'IMAGE(X)` -- string representation of X that is of discrete type T
- `T'VALUE(X)` -- value of discrete type T converted from the string X
- `T'POS(X)` -- integer position number of value of X of discrete type T
- `T'VAL(X)` -- value of discrete type T at position number X
- `T'SUCC(X)` -- value of discrete type T at position number X+1
- `T'PRED(X)` -- value of discrete type T at position number X+1
- `T'LEFTOF(X)` -- value of discrete type T at position left of X
- `T'RIGHTOF(X)` -- value of discrete type T at position right of X

### Array Indexes for an Array A

For multi-dimensional array, Nth index must be indicated in the attribute specifier. N can be omitted for a one-dimensional array.

- `A'LEFT` -- leftmost subscript of array A or constrained array type
- `A'RIGHT` -- rightmost subscript of array A or constrained array type
- `A'RIGHT(N)` -- rightmost subscript of dimension N of array A
- `A'HIGH` -- highest subscript of array A or constrained array type
- `A'HIGH(N)` -- highest subscript of dimension N of array A
- `A'LOW` -- lowest subscript of array A or constrained array type
- `A'LENGTH` -- integer value of the number of elements in array A
- `A'LENGTH(N)` -- number of elements of dimension N of array A
- `A'ASCENDING` -- boolean True if range of array A is defined with to
- `A'ASCENDING(N)` -- boolean True if range of array A is defined with to
### 6.10 Reserved Words

The following words are reserved for the VHDL-AMS language, so they cannot be used as identifiers. Reserved words are printed in bold in this Tutorial.

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Operator, absolute value of right operand.</td>
</tr>
<tr>
<td>ACCESS</td>
<td>Declares a type for creating access objects, pointers.</td>
</tr>
<tr>
<td>ACROSS</td>
<td>Declares the across quantity of a particular nature type.</td>
</tr>
<tr>
<td>AFTER</td>
<td>Defines a delay value (time after NOW) in signal assignment statements.</td>
</tr>
<tr>
<td>ALIAS</td>
<td>Defines an alternate name for an object.</td>
</tr>
<tr>
<td>ALL</td>
<td>Makes all items in a package visible, refers to all names of a class, or refers to all instances of a component.</td>
</tr>
<tr>
<td>AND</td>
<td>Operator, logical AND of left and right operands.</td>
</tr>
<tr>
<td>ARCHITECTURE</td>
<td>Primary design unit. Defines an architecture.</td>
</tr>
<tr>
<td>ARRAY</td>
<td>Declares an array data type containing a collection of elements that are all of the same type (array, vector, or matrix).</td>
</tr>
<tr>
<td>ASSERT</td>
<td>Checks a condition (occurrence of an event) and provides a report if the condition is not TRUE.</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>Defines an attribute name and its type.</td>
</tr>
<tr>
<td>BEGIN</td>
<td>Defines the begin of a block, entity, architecture, IF, or process statement.</td>
</tr>
<tr>
<td>BLOCK</td>
<td>Starts the description of a block structure.</td>
</tr>
<tr>
<td>BODY</td>
<td>Starts the description of various subprograms that are declared by the package body’s associated package declaration.</td>
</tr>
<tr>
<td>BUFFER</td>
<td>Port mode. Indicates a port which can be used for both input and output, and it can have only one source.</td>
</tr>
<tr>
<td>BUS</td>
<td>Signal mode. Defines a bus that floats to a user-specified value when all of its drivers are turned off.</td>
</tr>
<tr>
<td>CASE</td>
<td>Sequential statement. Executes one of several sequences of statements within a process, procedure, or function structure, depending on the value of a single expression.</td>
</tr>
<tr>
<td>COMPONENT</td>
<td>Starts a component declaration.</td>
</tr>
<tr>
<td>CONFIGURATION</td>
<td>Primary design unit. Defines a configuration for an entity.</td>
</tr>
<tr>
<td>CONSTANT</td>
<td>Declares an identifier name for a constant value (read only).</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>Signal driver condition. Defines the time delay to disconnect the guarded feature of a signal which is part of a guarded signal statement.</td>
</tr>
<tr>
<td>DOWNTO</td>
<td>Defines a descending range in a RANGE statement or other statement which includes a range.</td>
</tr>
<tr>
<td>ELSE</td>
<td>Defines the final alternative in an IF or WHEN statement.</td>
</tr>
<tr>
<td>ELSIF</td>
<td>Defines an interim alternative in an IF statement.</td>
</tr>
<tr>
<td>END</td>
<td>Defines the end of an entity, architecture, configuration, package, package body, or process statement.</td>
</tr>
<tr>
<td>Word</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>ENTITY</td>
<td>Primary design unit. Declares the input and output ports of a design.</td>
</tr>
<tr>
<td>EXIT</td>
<td>Sequential statement. Causes the immediate termination of a loop.</td>
</tr>
<tr>
<td>FILE</td>
<td>Declares a type for creating file handles and an identifier as a file object.</td>
</tr>
<tr>
<td>FOR</td>
<td>Identifies a parameter specification in a LOOP statement or time expression in a WAIT statement.</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>Defines a group of sequential statements that are executed when the function is called.</td>
</tr>
<tr>
<td>GENERATE</td>
<td>Copies a set of concurrent statements, or executes a set of concurrent statements selectively if a specified condition is met.</td>
</tr>
<tr>
<td>GENERIC</td>
<td>Specifies static information to be communicated to a model from its environment for all architectures.</td>
</tr>
<tr>
<td>GROUP</td>
<td>Defines a group template or specific group that can get an attribute.</td>
</tr>
<tr>
<td>GUARDED</td>
<td>Limits the execution of a signal statement. Causes a WAIT until a signal changes from FALSE to TRUE.</td>
</tr>
<tr>
<td>IF</td>
<td>Sequential statement. Performs a sequence of statements dependent on a defined condition.</td>
</tr>
<tr>
<td>IMPURE</td>
<td>Declares a function that is assumed to have side effects (return a different value given the same actual parameters).</td>
</tr>
<tr>
<td>IN</td>
<td>Port mode. Indicates a port which can be used for input.</td>
</tr>
<tr>
<td>INERTIAL</td>
<td>Clause in delay mechanism, followed from a time. Signals smaller than delay time are suppressed.</td>
</tr>
<tr>
<td>INOUT</td>
<td>Port mode. Indicates a port which can be used for both input and output.</td>
</tr>
<tr>
<td>IS</td>
<td>Used as a connective in various statements.</td>
</tr>
<tr>
<td>LABEL</td>
<td>Defines a label name in attribute statements.</td>
</tr>
<tr>
<td>LIBRARY</td>
<td>Designates a simple library name to identify libraries from which design units can be referenced.</td>
</tr>
<tr>
<td>LINKAGE</td>
<td>Port mode. Indicates a port which can be used for both input and output, and it can only correspond to a signal.</td>
</tr>
<tr>
<td>LITERAL</td>
<td>Entity class. Specifies an entity in attribute statements.</td>
</tr>
<tr>
<td>LOOP</td>
<td>Sequential statement. Executes a series of sequential statements multiple times.</td>
</tr>
<tr>
<td>MAP</td>
<td>Associates values of constants or ports within a structure to constants and ports outside the structure.</td>
</tr>
<tr>
<td>MOD</td>
<td>Operator, left operand modulo right operand.</td>
</tr>
<tr>
<td>NAND</td>
<td>Operator, logical NAND of left and right operands.</td>
</tr>
<tr>
<td>NATURE</td>
<td>Defines the across and through types of an object with a particular energy domain (nature).</td>
</tr>
<tr>
<td>NEW</td>
<td>Creates an object of a specified type and return an access value that refers to the created object.</td>
</tr>
<tr>
<td>NEXT</td>
<td>Sequential statement. Cause the next iteration in a loop.</td>
</tr>
<tr>
<td>Word</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NOISE</td>
<td>Declares the source aspect of a noise source quantity (serves as a source in a frequency domain model) in a quantity declaration.</td>
</tr>
<tr>
<td>NOR</td>
<td>Operator, logical <strong>NOR</strong> of left and right operands.</td>
</tr>
<tr>
<td>NOT</td>
<td>Operator, complement of right operand.</td>
</tr>
<tr>
<td>NULL</td>
<td>Sequential statement. Specifies explicitly that no action is needed.</td>
</tr>
<tr>
<td>OF</td>
<td>Used as a connective in various statements.</td>
</tr>
<tr>
<td>ON</td>
<td>Used as a connective in various statements.</td>
</tr>
<tr>
<td>OPEN</td>
<td>Designates the initial file characteristic or indicates a port that is not connected to any signal.</td>
</tr>
<tr>
<td>OR</td>
<td>Operator, logical <strong>OR</strong> of left and right operands.</td>
</tr>
<tr>
<td>OTHERS</td>
<td>Defines all remaining elements for example in a case statement, a selected assignment, and attribute specification.</td>
</tr>
<tr>
<td>OUT</td>
<td>Port mode. Indicates a port which can be used for output.</td>
</tr>
<tr>
<td>PACKAGE</td>
<td>Primary design unit. Defines a package and package body.</td>
</tr>
<tr>
<td>PORT</td>
<td>Specifies dynamic information to be communicated to a model from its environment for all architectures.</td>
</tr>
<tr>
<td>POSTPONED</td>
<td>Declares a process as a postponed process. Postponed processes do not execute until the final simulation cycle at the currently modeled time.</td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>Starts the description of a group of sequential statements that are to be executed when the procedure is called.</td>
</tr>
<tr>
<td>PROCESS</td>
<td>Starts the description of a group of sequential statements and is considered to be a single concurrent statement within a VHDL-AMS architecture.</td>
</tr>
<tr>
<td>PURE</td>
<td>Declares a function that is assumed to have no side effects.</td>
</tr>
<tr>
<td>QUANTITY</td>
<td>Declares a port in an entity declaration or across, through, and reference types in a nature declaration.</td>
</tr>
<tr>
<td>RANGE</td>
<td>Defines a range constraint for a scalar type.</td>
</tr>
<tr>
<td>RECORD</td>
<td>* Defines a record type and its corresponding element types.</td>
</tr>
<tr>
<td>REFERENCE</td>
<td>Declares the reference quantity of a particular nature type.</td>
</tr>
<tr>
<td>REGISTER</td>
<td>Signal mode. Defines a storage register that retains its last driven value when all of its drivers are turned off.</td>
</tr>
<tr>
<td>REJECT</td>
<td>Clause in delay mechanism, followed from a time. Signals smaller than reject time are suppressed.</td>
</tr>
<tr>
<td>REM</td>
<td>Operator, remainder of left operand divided by right operand.</td>
</tr>
<tr>
<td>REPORT</td>
<td>Defines a text string that is displayed when the condition in an ASSERT statement is not <strong>TRUE</strong>.</td>
</tr>
<tr>
<td>RETURN</td>
<td>Sequential statement. Terminates a subprogram (procedure or function) and returns control to the calling object.</td>
</tr>
<tr>
<td>ROL</td>
<td>Operator, left operand rotated left by right operand.</td>
</tr>
<tr>
<td>ROR</td>
<td>Operator, left operand rotated right by right operand.</td>
</tr>
<tr>
<td>Word</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SELECT</td>
<td>Selects and assigns a value to a target signal from among a list of alter-</td>
</tr>
<tr>
<td></td>
<td>natives, based on the value of a given expression.</td>
</tr>
<tr>
<td>SEVERITY</td>
<td>Defines the type of the message in an ASSERT statement.</td>
</tr>
<tr>
<td>SHARED</td>
<td>Declares shared objects.</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>Declares an identifier as a signal object.</td>
</tr>
<tr>
<td>SLA</td>
<td>Operator, left operand shifted left arithmetic by right operand.</td>
</tr>
<tr>
<td>SLL</td>
<td>Operator, left operand shifted left logical by right operand.</td>
</tr>
<tr>
<td>SPECTRUM</td>
<td>Declares the source aspect of a spectral source quantity (serves as a source</td>
</tr>
<tr>
<td></td>
<td>in a noise domain model) in a quantity declaration.</td>
</tr>
<tr>
<td>SRA</td>
<td>Operator, left operand shifted right arithmetic by right operand.</td>
</tr>
<tr>
<td>SRL</td>
<td>Operator, left operand shifted right logical by right operand.</td>
</tr>
<tr>
<td>SUBNATURE</td>
<td>Declares a nature that is a subnature of an existing nature.</td>
</tr>
<tr>
<td>SUBTYPE</td>
<td>Declares a type that is a subtype of an existing type.</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>Declares a terminal object of a particular nature.</td>
</tr>
<tr>
<td>THEN</td>
<td>Defines the first choice in an IF statement when the condition is TRUE.</td>
</tr>
<tr>
<td>THROUGH</td>
<td>Declares the through quantity of a particular nature type.</td>
</tr>
<tr>
<td>TO</td>
<td>Defines an ascending range in a RANGE statement or other statement which</td>
</tr>
<tr>
<td></td>
<td>includes a range.</td>
</tr>
<tr>
<td>TOLERANCE</td>
<td>Declares a tolerance that can be applied to scalar quantities.</td>
</tr>
<tr>
<td>TRANSPORT</td>
<td>Clause in delay mechanism, followed from a time. Defines a non-inertial delay in a signal assignment statement.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Declares a type.</td>
</tr>
<tr>
<td>UNAFFECTED</td>
<td>Indicates in a conditional or selected signal assignment when the signal is not to be given a new value.</td>
</tr>
<tr>
<td>UNITS</td>
<td>Declares physical types.</td>
</tr>
<tr>
<td>UNTIL</td>
<td>Defines a condition to terminate a WAIT statement.</td>
</tr>
<tr>
<td>USE</td>
<td>Makes a package available to this design unit.</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>Declares an identifier as a variable object.</td>
</tr>
<tr>
<td>WAIT</td>
<td>Suspends temporarily a process until a specified time has passed, a speci-</td>
</tr>
<tr>
<td></td>
<td>fied condition is met, or an event occurs which affects one or more signals.</td>
</tr>
<tr>
<td>WHEN</td>
<td>Defines a condition during which an exit or next statement will be executed or defines a choice (or choices) within a CASE statement.</td>
</tr>
<tr>
<td>WHILE</td>
<td>Defines a condition during which a loop will be executed.</td>
</tr>
<tr>
<td>WITH</td>
<td>Defines an expression in a selected signal assignment statement.</td>
</tr>
<tr>
<td>XOR</td>
<td>Operator, logical exclusive NOR of left and right operands.</td>
</tr>
<tr>
<td>XOR</td>
<td>Operator, logical exclusive OR of left and right operands.</td>
</tr>
</tbody>
</table>

* Partially supported in SIMPLORER 7.0.
** Not supported in SIMPLORER 7.0.
6.11 Modeling Aspects in SIMPLORER

Processes

A process statement defines an independent sequential process and is considered to be a single concurrent statement within a VHDL-AMS architecture. Process statements contain sequential statements but are themselves concurrent statements. They are the primary means of defining sequential statements. A process statement can include all declarations and sequential statements.

Quantities, Signals, and Variables

<table>
<thead>
<tr>
<th>Description</th>
<th>Direction</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantities</td>
<td>Analog objects</td>
<td>IN OUT =&gt; =&gt;</td>
</tr>
<tr>
<td>Signals</td>
<td>Digital objects</td>
<td>IN OUT INOUT &lt;=</td>
</tr>
<tr>
<td>Variables</td>
<td>Auxiliary objects to store intermediate values</td>
<td>None :=</td>
</tr>
</tbody>
</table>

Signal Assignments with Delay

VHDL-AMS offers several variants to perform signal assignments. The following example shows three delay forms:

ENTITY sequential_sig_assign IS
PORT(SIGNAL output: out bit);
END;

ARCHITECTURE behav OF sequential_sig_assign IS
SIGNAL sig_s,sig_t,sig_i,sig_r : bit;
BEGIN
  sig_s <= '1' AFTER 1 ms, '0' AFTER 5 ms,
           '1' AFTER 10 ms, '0' AFTER 13 ms,
           '1' AFTER 18 ms, '0' AFTER 20 ms,
           '1' AFTER 25 ms, '0' AFTER 26 ms;
PROCESS (sig_s)
BEGIN
  sig_t <= TRANSPORT sig_s AFTER 3 ms; -- signal assignment 1:1
  sig_i <= INERTIAL sig_s AFTER 3 ms;
  -- signal assignment without signals smaller than delay time are supressed
  sig_r <= REJECT 2 ms INERTIAL sig_s AFTER 3 ms;
  -- Signal assignment without signals smaller than reject time are supressed
END PROCESS;
END;
The following figure shows the results of the previous modeling code. The signal \( \text{sig}_s \) is delayed and assigned to signals in three different ways:

- The signal \( \text{sig}_t \) represents \( \text{sig}_s \) with a delay of 3ms.
- The signal \( \text{sig}_i \) represents \( \text{sig}_s \) with a delay of 3ms without signal changes occurring in an interval smaller than delay time.
- The signal \( \text{sig}_r \) represents \( \text{sig}_s \) with a delay of 3ms without signal changes occurring in an interval smaller than reject time.

---

### Data Exchange in Mixed-Signal Models

Mixed-signal models use analog and digital statements in their model description. Mixed-signal assignments should be avoided when other modeling variants can be used. Compiler errors or simulator instabilities can occur when data assignments are not performed in the correct way.

The following sections show examples of assignments from digital to analog values and analog to digital values. \text{Quantity}_\text{name} stands for an analog value, \text{signal}_\text{name} for a digital value.

#### Signal to Quantity Assignment (Digital to Analog)

The first example shows a value assigned in the correct syntax but without any support for the solver to find a good solution.

\[
\text{quantity}_\text{name} == \text{signal}_\text{name}; \text{syntax correct but solver problems can occur}
\]

To improve the solver stability, use a `\texttt{RAMP}` or `\texttt{SLEW}` attribute or a \texttt{BREAK} statement. The `\texttt{RAMP}` and `\texttt{SLEW}` attributes, as well as the \texttt{BREAK} statement, force a synchronization on the minimum simulator time step \( \text{HMIN} \) and thus minimize the error deviation of the solver.
The following examples show value assignments in connection with variants to define rise/fall time and positive/negative slew rate of a digital signal. The rise and fall time as well as positive and negative slew rate should be chosen so as to avoid infinite values of the first derivative for these quantities since this can cause simulator instabilities.

```
quantity_name == signal_name'RAMP(1.0e-9, 2.0e-9); -- define rise and fall time
quantity_name == signal_name'SLEW(100.0, 200.0); -- limit pos and neg slew rate
```

The following examples show value assignments dependent on events applied to a signal.

```
BREAK ON signal_name; -- assignment when event on signal_name
BREAK quantity_name'DOT => 1000.0 ON signal_name
BREAK qv => -qv WHEN NOT quantity_'Above(0.0);
quantity_name==signal_name;
```

**Quantity to Signal Assignment (Analog to Digital)**

The first example shows an incorrect way of using a value assignment. The statement is performed only during initialization, and the signal value is unchanged during rest of simulation. Signal assignments are characterized by events, and in this case quantities do not create any events.

```
signal_name <= quantity_name; -- only at init step, wrong assignment
```

The following example shows a value assignment if the quantity crosses a threshold value of 3.0. The signal value is of type BOOLEAN since the 'ABOVE attribute returns a TRUE or FALSE. In the next process, the actual value of the quantity is assigned to the signal. The next value assignment follows only after the quantity crosses the threshold again.

```
signal_name <= quantity_name'ABOVE(3.0); -- value of signal_name is true or false
PROCESS(signal_name)
BEGIN
  signal_name <= quantity_name; -- value of signal_name is actual value + maximum hmin
END;
```

The following example shows a value assignment at each time interval specified as delay:

```
clock <= NOT clock AFTER 1ms; -- generates an event at each time step defined as delay
PROCESS(clock)
BEGIN
  signal_name <= quantity_name;
END;
```
Solvability

The analog solver needs a specific set of equations when it evaluates the statements in the model. See also “DC-DC Converter” on page 42.

- The number of equations specified in the model must be equal to the sum of the number of free quantities, through quantities, and port quantities of mode `OUT`.
- All free quantities and port quantities of mode `OUT` must appear in a simultaneous equation within the architecture of the model.

The following example uses two relevant quantities: `I` (port quantity of mode `OUT`) and `ct` through quantity. Both quantities occur in the equations of the model.

```vhdl
LIBRARY IEEE;
USE IEEE.ELECTRICAL_SYSTEMS.ALL;
ENTITY AM IS
  PORT (QUANTITY I : OUT REAL; -- port quantity of mode out
         TERMINAL p,m : ELECTRICAL);
END ENTITY AM;
ARCHITECTURE behav OF AM IS
  QUANTITY v ACROSS ct THROUGH p TO m; -- through quantity ct
BEGIN
  I == ct; -- first equation with port and through quantity
  v == 0.0; -- second equation
END behav;
```

The digital solver needs a minimum simulation step `HMIN` that is less than or equal to the least delay specified with an `AFTER` statement. See also “DC-DC Converter” on page 42 and “Parameters of VHDL-AMS Simulator” on page 163.

**WORK Library**

Packages and models of a library are always compiled in the current WORK library. All packages and models of that library are visible to other models of the same library and can be instantiated as components in these models. If a model that instantiates models of the WORK library is copied to another library, the WORK library of this other library is used. That means, the same instantiation uses different models, because the WORK library was changed.

**Library 1**

```
LIBRARY WORK;
USE WORK.ALL;
ARCHITECTURE behav OF RC IS
R1: ENTITY WORK.R(behav)
  --R from library 1 is used
C2: ENTITY WORK.C(behav)
  --C from library 1 is used
```

**Library 2**

```
LIBRARY WORK;
USE WORK.ALL;
ARCHITECTURE behav OF RC IS
R1: ENTITY WORK.R(behav)
  --R from library 2 is used
C2: ENTITY WORK.C(behav)
  --C from library 2 is used
```

Models defined in VHDL-AMS text subsheets that are placed on a model sheet are also compiled in the current sheet’s WORK library. All models of these subsheets are visible to other subsheets of the same sheet at any hierarchy level.
Entity names defined in subsheets must be unique within the entire sheet, because an entity with a duplicate name will overwrite the previously defined one.

The VHDL-AMS working directory of SIMPLORER is...

Alias for File Names

A file alias name can be defined for a library name in the SIMPLROER SSC commander. Choose OPTIONS>PROGRAM DIRECTORIES and click the «Alias» tab. Create a new file alias and define alias name and path of the corresponding file. To use files with names that do not conform to VHDL-AMS identifier rules, such as a file name with spaces, specify a file alias.
Values on Sheet

In the SIMPLORER Schematic, parameter values for VHDL-AMS models are entered in component dialogs. Values can be assigned to each parameter corresponding to the inputs defined in the model entity. If no value is entered, always the default value specified in the model is used. See also Chapter 3, “Automotive Powernet System Example” on page 25.

The syntax of user-defined values must conform to the SML. VHDL-AMS attributes and syntax cannot be used. Each value assigned to a parameter is of type `REAL`. The following table lists possible value assignments for VHDL-AMS data types and their effects. See also Chapter A.3, “Common Conventions in SML” on page 218.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Default</th>
<th>Data Type</th>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inp1</td>
<td>-1</td>
<td>-1.0</td>
<td>REAL</td>
<td>SIGNAL</td>
<td>All values are possible. If no default value is specified, name'LEFT is used (-1.7e-308).</td>
</tr>
<tr>
<td>inp2</td>
<td>var1</td>
<td>0.0</td>
<td>REAL</td>
<td>QUANTITY</td>
<td></td>
</tr>
<tr>
<td>inp3</td>
<td>ctrl</td>
<td>'1'</td>
<td>BIT</td>
<td>SIGNAL</td>
<td>Values between 0 and 2 are possible, otherwise an error message occurs during simulation.</td>
</tr>
<tr>
<td>inp4</td>
<td>0</td>
<td>inp3left</td>
<td>BIT</td>
<td>GENERIC</td>
<td>• if inp&lt;1 then use value 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• if inp&gt;=1 then use value 1</td>
</tr>
<tr>
<td>inp5</td>
<td>2.5</td>
<td>10</td>
<td>INTEGER</td>
<td>SIGNAL</td>
<td>All values are possible. If inp is not integer value, the integer part of the value is used.</td>
</tr>
<tr>
<td>inp6</td>
<td>var2</td>
<td>0</td>
<td>INTEGER</td>
<td>GENERIC</td>
<td></td>
</tr>
<tr>
<td>inp7</td>
<td>7</td>
<td>'H'</td>
<td>STD_LOGIC</td>
<td>SIGNAL</td>
<td>Values between 0 and 8 are possible, otherwise an error message occurs during simulation.</td>
</tr>
<tr>
<td>inp8</td>
<td>var3</td>
<td>'0'</td>
<td>STD_LOGIC</td>
<td>SIGNAL</td>
<td></td>
</tr>
</tbody>
</table>

In addition to value specifications in the model dialog, OmniCaster models can be used to perform value assignments. OmniCasters convert different data types in the correct way if a conversion is possible, otherwise they reject a connection between the parameters. See also “Using Transformation Models” on page 18.

Vector Inputs on Sheet

Vector inputs appear in the component input dialog as parameters with index. Values for dynamic vectors can only be assigned with a wire.

```vhdl
ENTITY DAC IS  
  GENERIC (MIN: REAL := -1.0; MAX: REAL := 1.0);  
  PORT (  
    SIGNAL INPUT: IN BIT_VECTOR(3 DOWNTO 0) := (OTHERS => '0');  
    QUANTITY VAL: OUT REAL := 0.0);  
END ENTITY DAC;  
```
## A Appendix

### A.1 SIMPLER Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC simulation</td>
<td>Harmonic analysis of a model.</td>
</tr>
<tr>
<td>AC simulator</td>
<td>SIMPLER simulator for AC analysis.</td>
</tr>
<tr>
<td>Action type</td>
<td>Defines how an action in a state is processed.</td>
</tr>
<tr>
<td>.afa file</td>
<td>Analytical Frequency Analysis file containing data of an analytical frequency analysis.</td>
</tr>
<tr>
<td>Analytic frequency step</td>
<td>SIMPLER module to compute frequency step response information for a given transfer function.</td>
</tr>
<tr>
<td>response</td>
<td></td>
</tr>
<tr>
<td>Animated Symbol</td>
<td>Symbol for a component or a macro that changes to reflect changes in values assigned to it. The symbol can be modified by the user with an interaction button or by a system value during the simulation. Animated symbols can be generated using the Symbol Editor.</td>
</tr>
<tr>
<td>.aws file</td>
<td>Simulator file containing initial values of capacitors and inductors.</td>
</tr>
<tr>
<td>Backplane</td>
<td>Common communication and control structure for all SIMPLER simulators.</td>
</tr>
<tr>
<td>Basic version</td>
<td>SIMPLER basic version without any optional modules.</td>
</tr>
<tr>
<td>Basics</td>
<td>SIMPLER library containing the basic modeling elements for the SIMPLER simulators.</td>
</tr>
<tr>
<td>Behavioral model</td>
<td>Smallest model unit that cannot be subdivided further.</td>
</tr>
<tr>
<td>Best representation</td>
<td>Function for scaling all output channels of a graph window to maximum size.</td>
</tr>
<tr>
<td>Bezier</td>
<td>Bezier curves are used in computer graphics to produce curves which appear reasonably smooth at all scales. The curves are constructed as a sequence of cubic segments, rather than linear ones. Bezier curves use a construction in which the interpolating polynomials depend on certain control points. The mathematics of these curves is classical, but it was a French automobile engineer Pierre Bezier who introduced their use in computer graphics.</td>
</tr>
<tr>
<td>Block</td>
<td>Linear or nonlinear transfer function, basic element of block diagrams.</td>
</tr>
<tr>
<td>Block diagram</td>
<td>Combination of blocks to describe the dynamic behavior of systems.</td>
</tr>
<tr>
<td>Block Diagram Module (BDM)</td>
<td>SIMPLER sub simulator to analyze simulation models described using block diagrams. Uses Euler formula and distributed integration algorithms.</td>
</tr>
<tr>
<td>Bookmark</td>
<td>Can be used in the SIMPLER editor to mark a position and find it again.</td>
</tr>
<tr>
<td>.brs file</td>
<td>External Schematic file to parametrize a model sheet.</td>
</tr>
<tr>
<td>Characteristic</td>
<td>Function or data set to describe a nonlinear characteristic of a SIMPLER component, block or other model.</td>
</tr>
<tr>
<td>Characteristic values</td>
<td>Analysis method in the DAY Post Processor that provides a list of frequently used characteristic values of a system quantity.</td>
</tr>
<tr>
<td>C-Interface</td>
<td>C/C++ programming interface for the integration of user defined nonlinear models or components into the simulation model.</td>
</tr>
<tr>
<td>Color scheme</td>
<td>Predefined or user defined color settings for screen outputs or printing.</td>
</tr>
<tr>
<td>Compiler</td>
<td>Program for the translation of the SML description of a simulation model into a simulator specific format.</td>
</tr>
<tr>
<td>Computation sequence</td>
<td>Sequence in which blocks in a simulation model are computed.</td>
</tr>
<tr>
<td>Connection rule</td>
<td>Rule determining which element can be connected to another under certain conditions.</td>
</tr>
<tr>
<td>Conservative node</td>
<td>Connection of two or more circuit component terminals.</td>
</tr>
<tr>
<td>Coordinate system</td>
<td>Reference system for the display of numerical values. It can be linear or logarithmic.</td>
</tr>
<tr>
<td>Crossing over</td>
<td>Exchange of genetic material between chromosomes in a genetic algorithm.</td>
</tr>
<tr>
<td>Cursor</td>
<td>Positioning element to determine the value of a quantity in a coordinate system.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Data cache</td>
<td>Saves the data of the last simulation run.</td>
</tr>
<tr>
<td>Data channel</td>
<td>Simulation data for a specific quantity stored in a file or transferred to an active element.</td>
</tr>
<tr>
<td>Data filter</td>
<td>Function to select data by user defined criteria.</td>
</tr>
<tr>
<td>Data format</td>
<td>Defines how data are stored or exchanged between programs.</td>
</tr>
<tr>
<td>Data set</td>
<td>Simulation data of a simulation run at a given time step.</td>
</tr>
<tr>
<td>day file</td>
<td>DAY Post Processor file containing data of a data analysis.</td>
</tr>
<tr>
<td>DAY Optim</td>
<td>SIMPLORER module for the evaluation of optimization results.</td>
</tr>
<tr>
<td>DAY Postprocessor</td>
<td>SIMPLORER module for the post processing of simulation and measurement data.</td>
</tr>
<tr>
<td>DC simulation</td>
<td>DC operating point analysis of a model.</td>
</tr>
<tr>
<td>DC simulator</td>
<td>SIMPLORER simulator for DC analysis.</td>
</tr>
<tr>
<td>Differentiation</td>
<td>Analysis method in the DAY Post Processor.</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>SIMPLORER library containing display elements to display simulation data inside a schematic.</td>
</tr>
<tr>
<td>Display Element</td>
<td>Element for graphic or numeric online visualization of simulation data on screen.</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic link library containing program components or models. It is loaded automatically upon request.</td>
</tr>
<tr>
<td>Dongle</td>
<td>Software protection device connected to the printer port of the computer.</td>
</tr>
<tr>
<td>DSDE</td>
<td>Dynamic Simulation Data Exchange interface used to integrate external simulators on a client-server level.</td>
</tr>
<tr>
<td>Electric circuit</td>
<td>Combination of electric components, connected by ideal wires.</td>
</tr>
<tr>
<td>Electric Circuit Module (ECM)</td>
<td>SIMPLORER sub simulator to analyze simulation models described using electric circuits. Uses Euler or Trapezoid algorithm and modified nodal approach.</td>
</tr>
<tr>
<td>Element</td>
<td>Any item that can be placed on a Schematic sheet, including models, text elements, display elements, drawing elements, sub sheets, and so on.</td>
</tr>
<tr>
<td>Entity</td>
<td>VHDL-AMS term to describe the interface of a behavioral or structural model.</td>
</tr>
<tr>
<td>Euler formula</td>
<td>Numerical integration algorithm used inside SIMPLORER.</td>
</tr>
<tr>
<td>Evaluation function</td>
<td>Function evaluating the quality of a solution compared to the defined optimum.</td>
</tr>
<tr>
<td>.exp file</td>
<td>File containing the settings of an experiment.</td>
</tr>
<tr>
<td>Experiment tool</td>
<td>SIMPLORER module for the definition and management of experiments performed using a model.</td>
</tr>
<tr>
<td>Experiment wizard</td>
<td>Special wizard for easy definition of experiments.</td>
</tr>
<tr>
<td>Expert mode</td>
<td>Extended mode for the definition of an experiment. It allows fine tuning of parameters and methods.</td>
</tr>
<tr>
<td>Export filter</td>
<td>Special program for the export of simulation data into other applications.</td>
</tr>
<tr>
<td>Extern View</td>
<td>Special sheet independent oscilloscope for the display of simulation data.</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transformation.</td>
</tr>
<tr>
<td>.fil file</td>
<td>DAY Optim file containing data of a filter applied to the data file.</td>
</tr>
<tr>
<td>File output</td>
<td>Saves a system quantity online during the simulation in a file.</td>
</tr>
<tr>
<td>Fitness function</td>
<td>Function describing the fitness of an individual (parameter constellation) in a genetic algorithm.</td>
</tr>
<tr>
<td>Formula Module (FML)</td>
<td>SIMPLORER’s integrated expression evaluator.</td>
</tr>
<tr>
<td>Frequency step response analysis</td>
<td>Special mode in the Experiment tool, determining the frequency behavior of a simulation model.</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>Optimization method of the experiment tool with automatic parameter variation and target function determination.</td>
</tr>
<tr>
<td>Grid lines</td>
<td>Grid lines in a coordinate system.</td>
</tr>
<tr>
<td>HMAX</td>
<td>Simulation parameter, maximum step size for the integration algorithm.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HMIN</td>
<td>Simulation parameter, minimum step size for the integration algorithm.</td>
</tr>
<tr>
<td>ID</td>
<td>Numeric value for the distinction of sheet elements.</td>
</tr>
<tr>
<td>Information window</td>
<td>Display of warnings, errors, program status for the SIMPLORER modules.</td>
</tr>
<tr>
<td>Initial condition</td>
<td>Defines the initial value for energy storing electrical components.</td>
</tr>
<tr>
<td>Initial state</td>
<td>A state that is active at the beginning of the simulation.</td>
</tr>
<tr>
<td>INT</td>
<td>Simulation parameter, defines the integration algorithm used for the simulation of electric circuits.</td>
</tr>
<tr>
<td>Interactivity pad</td>
<td>Part of an animated symbol, used to change the behavior and/or shape of a symbol (model component) by the user.</td>
</tr>
<tr>
<td>Iteration</td>
<td>Part of the integration process for the solution of nonlinear problems.</td>
</tr>
<tr>
<td>Jacobian Matrix</td>
<td>Coefficient matrix for the numerical integration algorithm.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Can be used to do a search in a model database.</td>
</tr>
<tr>
<td>.krn file</td>
<td>Simulator file containing simulation status information. It can be used as a starting point for the next simulation.</td>
</tr>
<tr>
<td>Language concept</td>
<td>Settings to define the use of language for program (menus and dialogs) and libraries.</td>
</tr>
<tr>
<td>Library</td>
<td>Database containing a set of SIMPLORER basic elements and/or macro models.</td>
</tr>
<tr>
<td>Local discretization error</td>
<td>Simulation parameter, determines the accuracy of the computation according to the dynamics of the electric circuits.</td>
</tr>
<tr>
<td>Log file</td>
<td>Experiment tool file containing the protocol of an experiment.</td>
</tr>
<tr>
<td>Macro</td>
<td>Contains one or more elements of a model description that can be used as single elements.</td>
</tr>
<tr>
<td>Main skeleton (.skl)</td>
<td>Special file containing basic descriptions for the generation of the model description file. Must not be modified.</td>
</tr>
<tr>
<td>Maximum current error</td>
<td>Simulation parameter that determines the accuracy of the right side computation of the differential equation system for current lines.</td>
</tr>
<tr>
<td>Maximum number of iterations</td>
<td>Simulation parameter that limits the number of iteration loops for the nonlinear iteration process.</td>
</tr>
<tr>
<td>Maximum voltage error</td>
<td>Simulation parameter that determines the accuracy of the right side computation of the differential equation system for voltage lines.</td>
</tr>
<tr>
<td>.mda/.mdk file</td>
<td>SIMPLORER data file of simulation results and characteristics (former format).</td>
</tr>
<tr>
<td>.mdx file</td>
<td>SIMPLORER data file of simulation results and characteristics.</td>
</tr>
<tr>
<td>Model</td>
<td>All elements in the libraries and subsheets (except for display elements).</td>
</tr>
<tr>
<td>Model tree</td>
<td>Hierarchical list box containing the available elements for the graphical modeling.</td>
</tr>
<tr>
<td>ModelAgent</td>
<td>SIMPLORER module for the model library management.</td>
</tr>
<tr>
<td>Module</td>
<td>SIMPLORER sub simulator or program.</td>
</tr>
<tr>
<td>Monitor</td>
<td>Special window for the display of simulation status and progress.</td>
</tr>
<tr>
<td>Monte Carlo Analysis</td>
<td>Optimization method of the experiment tool with automatic parameter generation and characteristic value determination.</td>
</tr>
<tr>
<td>.mtx file</td>
<td>SIMPLORER data file of the DES model.</td>
</tr>
<tr>
<td>Multi simulation</td>
<td>Analysis method of the experiment tool, batch mode computation of a simulation model with different, user defined parameter sets.</td>
</tr>
<tr>
<td>Mutation</td>
<td>Part of the genetic algorithm that generates new individuals (parameter sets) by randomly modified parameters.</td>
</tr>
<tr>
<td>Network installation</td>
<td>Installation process of a SIMPLORER network version.</td>
</tr>
<tr>
<td>Newton-Raphson-Algorithm</td>
<td>Nonlinear iteration algorithm used in SIMPLORER.</td>
</tr>
<tr>
<td>Non-conservative node</td>
<td>Connection of two or more non-circuit component terminals.</td>
</tr>
<tr>
<td>Normal mode</td>
<td>Default display of the experiment tool without evaluation functions and storage options.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Object</td>
<td>Element of the graphic model description linked to another WINDOWS application (OLE).</td>
</tr>
<tr>
<td>Object browser</td>
<td>Special window for the display and browsing of the elements of a graphical model description.</td>
</tr>
<tr>
<td>Offline</td>
<td>When the simulation is not running in SIMPLORER.</td>
</tr>
<tr>
<td>Online</td>
<td>During the simulation in SIMPLORER.</td>
</tr>
<tr>
<td>Online graphic</td>
<td>Online display of simulation results during the simulation run.</td>
</tr>
<tr>
<td>Online graphic output</td>
<td>Display of a system quantity during the simulation in a Display element or the ViewTool.</td>
</tr>
<tr>
<td>Option</td>
<td>Optional SIMPLORER application. It can be registered using the installation manager.</td>
</tr>
<tr>
<td>Output</td>
<td>Definition of a specific quantity of the simulation model to be used as an output.</td>
</tr>
<tr>
<td>Password</td>
<td>Letter and digit combination to access SIMPLORER.</td>
</tr>
<tr>
<td>Petri net</td>
<td>Special form of a state machine.</td>
</tr>
<tr>
<td>Pipe</td>
<td>Data channel for the data transmission between SIMPLORER modules.</td>
</tr>
<tr>
<td>Postprocessing</td>
<td>Data evaluation and processing after a simulation run has finished.</td>
</tr>
<tr>
<td>Power</td>
<td>Analysis method in the DAY Post Processor.</td>
</tr>
<tr>
<td>Preprocess</td>
<td>Specialized modules for information processing and parameter determination to define SIMPLORER model components.</td>
</tr>
<tr>
<td>Preprocessor directive</td>
<td>Special commands for the SML compiler to include files, extract macros from the model library, etc.</td>
</tr>
<tr>
<td>Presentation</td>
<td>Special mode of the DAY Post Processor. It allows the arrangement of results in a reusable form.</td>
</tr>
<tr>
<td>Print colors</td>
<td>Colors used to print a system quantity from an active element or the ViewTool.</td>
</tr>
<tr>
<td>Project</td>
<td>Organizational structure containing all files and information belonging to a simulation task.</td>
</tr>
<tr>
<td>Qualifier</td>
<td>References a system quantity or parameter of a SIMPLORER model component.</td>
</tr>
<tr>
<td>Quality criterion</td>
<td>Characteristic value of a system quantity used to determine the quality of an optimization run.</td>
</tr>
<tr>
<td>Queue</td>
<td>Display the active and all waiting simulation runs.</td>
</tr>
<tr>
<td>Recombination</td>
<td>Part of the genetic algorithm. It generates new individuals (parameter sets) by crossing of two parent individuals.</td>
</tr>
<tr>
<td>Roll back</td>
<td>Go back to a previous simulation step.</td>
</tr>
<tr>
<td>Sample time</td>
<td>Step size for digital controller.</td>
</tr>
<tr>
<td>Schematic</td>
<td>SIMPLORER module for the graphical model definition.</td>
</tr>
<tr>
<td>Screen colors</td>
<td>Colors used to display a system quantity online during the simulation in an active element or the ViewTool.</td>
</tr>
<tr>
<td>Section</td>
<td>Part of the SML description containing model information for the SIMPLORER sub simulators.</td>
</tr>
<tr>
<td>Selection</td>
<td>Part of the genetic algorithm. It selects individuals of the active generation to be transferred to the next generation by specific selection criteria.</td>
</tr>
<tr>
<td>SIMPLORER program</td>
<td>SIMPLORER application launched from the SSC commander.</td>
</tr>
<tr>
<td>SIMPLORER settings</td>
<td>Contains the settings for the program environment, paths, etc.</td>
</tr>
<tr>
<td>SIMPLORER tools</td>
<td>Additional programs included in the software package but not necessarily integrated into the SSC commander.</td>
</tr>
<tr>
<td>Simulation backplane technology</td>
<td>SIMPLORER software architecture for data exchange and program control of several simulators in one environment.</td>
</tr>
<tr>
<td>Simulation model</td>
<td>Graphical or text description of a real system using modeling capabilities of a simulation system.</td>
</tr>
<tr>
<td>Simulation parameter</td>
<td>Parameter used to control the simulation process.</td>
</tr>
<tr>
<td>Simulator</td>
<td>Software for the analysis of the behavior of a system using a simulation model.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Simulator coupling</td>
<td>Direct link of one or more simulators using the SIMPLORER simulation back-</td>
</tr>
<tr>
<td></td>
<td>plane technology.</td>
</tr>
<tr>
<td>Simulator interface</td>
<td>Special software interface for the integration of external simulators into SIM-</td>
</tr>
<tr>
<td></td>
<td>PLORER.</td>
</tr>
<tr>
<td>.smd file</td>
<td>SIMPLORER ModelAgent file containing data of a model library.</td>
</tr>
<tr>
<td>SML</td>
<td>SIMPLORER Modeling Language.</td>
</tr>
<tr>
<td>.sml file</td>
<td>File containing the model description of a simulation run.</td>
</tr>
<tr>
<td>SML key word</td>
<td>Special terms used to determine the sections of an SML description.</td>
</tr>
<tr>
<td>Smooth</td>
<td>Analysis method in the DAY Post Processor.</td>
</tr>
<tr>
<td>Solver</td>
<td>Algorithm for the computation of model components without the capability to</td>
</tr>
<tr>
<td></td>
<td>roll back steps.</td>
</tr>
<tr>
<td>SSC commander</td>
<td>SIMPLORER application manager, project manager and communication interface.</td>
</tr>
<tr>
<td>.asc file</td>
<td>SIMPLORER project file, containing all information about a project.</td>
</tr>
<tr>
<td>.ash file</td>
<td>SIMPLORER schematic file, containing the graphical representation of a model</td>
</tr>
<tr>
<td></td>
<td>and simulation data.</td>
</tr>
<tr>
<td>State</td>
<td>Basic element of state graphs that defines properties and activities in a cer-</td>
</tr>
<tr>
<td></td>
<td>tain system state.</td>
</tr>
<tr>
<td>State graph</td>
<td>Combination of states and transitions. A modeling language for discontinu-</td>
</tr>
<tr>
<td></td>
<td>ous systems.</td>
</tr>
<tr>
<td>State Graph Module (SGM)</td>
<td>SIMPLORER sub simulator to analyze simulation models described using state</td>
</tr>
<tr>
<td></td>
<td>graphs basing on the PETRI net theory.</td>
</tr>
<tr>
<td>Status line</td>
<td>Displays the present state of a SIMPLORER application.</td>
</tr>
<tr>
<td>Subsheet</td>
<td>A SIMPLORER sheet embedded into another SIMPLORER sheet, connected via pins,</td>
</tr>
<tr>
<td></td>
<td>automatically creates a macro inside the .sml file.</td>
</tr>
<tr>
<td>Sub-simulator</td>
<td>SIMPLORER internally coupled simulator.</td>
</tr>
<tr>
<td>Sub-skeleton</td>
<td>File containing rules for the creation of a non-SIMPLORER description lan-</td>
</tr>
<tr>
<td></td>
<td>guage.</td>
</tr>
<tr>
<td>Successive Approximation</td>
<td>Optimization method of the Experiment Tool with automatic parameter varia-</td>
</tr>
<tr>
<td></td>
<td>tion and target function determination.</td>
</tr>
<tr>
<td>Symbol editor</td>
<td>SIMPLORER application for the creation or modification of a symbol.</td>
</tr>
<tr>
<td>Symbol level</td>
<td>Group of drawing elements of an animated symbol displayed together depend-</td>
</tr>
<tr>
<td></td>
<td>ing on the input value or the user activity on a interaction pad.</td>
</tr>
<tr>
<td>Symbols</td>
<td>Graphic representations of elements on the Schematic, placed from the model</td>
</tr>
<tr>
<td></td>
<td>library. They can be modified using the symbol editor.</td>
</tr>
<tr>
<td>Synchronization</td>
<td>Update of a schematic from an older SIMPLORER version to the latest symbols.</td>
</tr>
<tr>
<td>Synchronization</td>
<td>Detection of events in a state graph.</td>
</tr>
<tr>
<td>System quantity</td>
<td>Any quantity computed by the simulator.</td>
</tr>
<tr>
<td>System simulation</td>
<td>Simulation level, where models from different physical domains are simulated</td>
</tr>
<tr>
<td></td>
<td>at the same time.</td>
</tr>
<tr>
<td>System variable</td>
<td>Predefined variables inside SIMPLORER that cannot be used as a variable name</td>
</tr>
<tr>
<td></td>
<td>or a specifier.</td>
</tr>
<tr>
<td>Task</td>
<td>Analysis to be performed as part of an experiment. It can contain several sim-</td>
</tr>
<tr>
<td></td>
<td>ulation runs.</td>
</tr>
<tr>
<td>Template</td>
<td>Predefined structure for a SIMPLORER application.</td>
</tr>
<tr>
<td>TEND</td>
<td>Simulation parameter that determines the simulation end time.</td>
</tr>
<tr>
<td>Text Editor</td>
<td>SIMPLORER application for the definition of models in SML language.</td>
</tr>
<tr>
<td>Time Function Module</td>
<td>SIMPLORER sub simulator for the computation of time dependent functions.</td>
</tr>
<tr>
<td>(TFM)</td>
<td></td>
</tr>
<tr>
<td>Time limited</td>
<td>A license of the simulation software for a certain (limited) time period.</td>
</tr>
<tr>
<td>Time step</td>
<td>Present time step size used to compute the next results vector.</td>
</tr>
<tr>
<td>Toolbar</td>
<td>Bar in a SIMPLORER application to perform activities in the application.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Total fitness</td>
<td>The total fitness of an optimization run.</td>
</tr>
<tr>
<td>TR simulation</td>
<td>Transient analysis of a model.</td>
</tr>
<tr>
<td>TR simulator</td>
<td>SIMPLORER simulator for TR analysis.</td>
</tr>
<tr>
<td>Transition</td>
<td>Cross over condition between the input and output state(s) of a state graph, defined by a logical expression.</td>
</tr>
<tr>
<td>Transition component</td>
<td>Basic structure of a state machine. It comprises a transition and all of its input and output states.</td>
</tr>
<tr>
<td>Trapezoid Formula</td>
<td>Numerical algorithm used inside SIMPLORER.</td>
</tr>
<tr>
<td>Trend Analysis</td>
<td>Analysis method of the Experiment Tool with automatic parameter variation and characteristic value determination.</td>
</tr>
<tr>
<td>UDMinit</td>
<td>Section of the C/C++ interface for the model initialization.</td>
</tr>
<tr>
<td>UDMMain</td>
<td>Section of the C/C++ interface containing the model computation algorithm.</td>
</tr>
<tr>
<td>User defined component (UDC)</td>
<td>Model description for electrical components using a user defined C/C++ program, with direct access to the solver matrix.</td>
</tr>
<tr>
<td>User defined model (UDM)</td>
<td>Model description for nonlinear characteristics using a user defined C/C++ program.</td>
</tr>
<tr>
<td>User management</td>
<td>Saves the workspace for individual users.</td>
</tr>
<tr>
<td>Version report</td>
<td>Special mode inside the SIMPLORER help system to create a detailed information file about the present SIMPLORER installation.</td>
</tr>
<tr>
<td>VHDL</td>
<td>Very high-speed integrated circuit Hardware Description Language for digital systems.</td>
</tr>
<tr>
<td>VHDL-AMS</td>
<td>Very high-speed integrated circuit Hardware Description Language - Analog Mixed Signal. Extension of VHDL, a hardware description language for digital and analog systems.</td>
</tr>
<tr>
<td>VHDL-AMS simulator</td>
<td>Simulator integrated with SIMPLORER’s backplane that calculates simulation models described in VHDL-AMS.</td>
</tr>
<tr>
<td>ViewTool</td>
<td>Program to display results online during the simulation outside the Schematic.</td>
</tr>
<tr>
<td>Window elements</td>
<td>Windows in the workspace containing various information. They can be turned on or off by the user.</td>
</tr>
<tr>
<td>Work flow</td>
<td>Graphic representation of a sequence of activities that are performed on a certain data set.</td>
</tr>
<tr>
<td>Worst Case Analysis</td>
<td>Analysis method of the Experiment Tool with parameter combination of all extreme values for the parameters and characteristic value determination.</td>
</tr>
<tr>
<td>YMAX</td>
<td>Maximum value used for display in the online graphic using the View tool.</td>
</tr>
<tr>
<td>YMIN</td>
<td>Minimum value used for display in the online graphic using the View tool.</td>
</tr>
</tbody>
</table>
### A.2 Table of SIMPLORER Libraries

<table>
<thead>
<tr>
<th>ModelAgent Tab</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basics</td>
<td>The «Basics» tab provides electric circuit components, blocks, states, measuring devices, signal characteristics (functions to evaluate characteristics online during simulation), modeling tools (time functions, characteristics, equations), and components of physical domains.</td>
</tr>
<tr>
<td>Displays</td>
<td>The «Displays» tab provides elements for visual online display of graphical and numerical simulation outputs during a simulation on the Schematic sheet.</td>
</tr>
<tr>
<td>AMS</td>
<td>The «AMS» tab provides electric circuit components, blocks, measuring devices, modeling tools (Time functions), and components of several physical domains modeled in VHDL-AMS.</td>
</tr>
<tr>
<td>Digital</td>
<td>The «Digital» tab provides components with common basic functionality used for simple digital circuits.</td>
</tr>
</tbody>
</table>
| Add Ons        | • Power Electronics Library (power.smd)  
                  • Switch Mode Power Supply Library (smps.smd)  
                  • Automotive Components Library (automotive.smd)  
                  • One-Dimensional Mechanical System Module (mechsim.smd)  
                  • Hydraulic Components Library (hydraulic.smd)  
                  • Advanced Machine Library (ist_machine.smd, servo_ist.smd)  
                  • Sensor Model Library (sensors.smd)  
                  • Specific Semiconductor Model Libraries (bjt.smd, diodes.smd, thyristors.smd)  
                  • Nonlinear transmission and linear transmission components (transf.smd)  
                  • Interfaces library (interfaces.smd)  
                  - ECE (equivalent circuit extraction) Maxwell interface  
                  - RMxprt interface  
                  - FExpt interface  
                  - FEA interface  
                  - Full-Wave SPICE interface  
                  - Mathcad interface  
                  - Matlab/Simulink interface |
| Tools          | The «Tools» tab provides a library of components that calculate coordinate transformations, connect conservative nodes from different natures, and connect different data types. A compatibility library that ensures backwards compatibility of models from previous versions is also available. |
| Manufacturers   | The «Manufacturers» tab provides libraries with semiconductor device level models of different manufactures. |
| Users          | The «Users» tab provides a location to insert user-defined libraries. |
| Projects       | The «Projects» tab provides libraries included in an opened project. |
A.3 Common Conventions in SML

This appendix refers only to conventions in SML, not VHDL-AMS.

Names of Components and Variables

User-defined names can be given to all components (internal components, C-models, macros, nodes, ports and variables). Names may consist of letters, digits, and underscores and can have a maximum of 50 characters.

Vowel mutations (umlauts) and spaces are not allowed. All names are case sensitive. The first character must always be a letter.

The following are not allowed for names:
- SML notation keywords
- Simulation parameters
- System variables

Unit Suffixes of Numeric Data

Numeric data can be entered in Schematic component dialogs and in the Text Editor, using the following unit extensions:

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Value</th>
<th>SML</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>tera</td>
<td>$10^{12}$</td>
<td>T</td>
<td>$5e12, 5t, 5ter$</td>
</tr>
<tr>
<td>giga</td>
<td>$10^{9}$</td>
<td>G</td>
<td>$1.4e9, 1.4g, 1.4giga$</td>
</tr>
<tr>
<td>mega</td>
<td>$10^{6}$</td>
<td>M</td>
<td>$-1.4E6, -0.3meg, -0.3MEG$</td>
</tr>
<tr>
<td>kilo</td>
<td>$10^{3}$</td>
<td>K</td>
<td>$1000, 1e3, 1k, 1kil$</td>
</tr>
<tr>
<td>milli</td>
<td>$10^{-3}$</td>
<td>m</td>
<td>$0.0105, 1.0E-2, 10.5M, 10.5MIL$</td>
</tr>
<tr>
<td>micro</td>
<td>$10^{-6}$</td>
<td>µ</td>
<td>$0.000005, 5e-6, 5u, 5mic$</td>
</tr>
<tr>
<td>nano</td>
<td>$10^{-9}$</td>
<td>n</td>
<td>$40E-9, 40n, 40nan$</td>
</tr>
<tr>
<td>pico</td>
<td>$10^{-12}$</td>
<td>p</td>
<td>$100E-12, 100P, 100PIC$</td>
</tr>
<tr>
<td>femto</td>
<td>$10^{-15}$</td>
<td>f</td>
<td>$9E-15, 9F, 9FEM$</td>
</tr>
</tbody>
</table>

The comma is reserved for separating parameters in lists. The period (dot) is reserved as a decimal point. "M" is interpreted as $10^{-3}$, not as $10^6$.

SI Units

All units used from internal simulator components are derived from the SI Units system.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Unit Name</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Meters</td>
<td>m</td>
</tr>
<tr>
<td>Mass</td>
<td>Kilograms</td>
<td>kg</td>
</tr>
<tr>
<td>Time</td>
<td>Seconds</td>
<td>s</td>
</tr>
</tbody>
</table>
Parameter Qualifiers

Components are characterized by different physical quantities. A resistor, for example, is represented by current and voltage in the simulation. System variables may be accessed by users for both reading (to use the actual quantity in expressions or to create an output) and writing (influencing quantities) with the following syntax:

```
Name.Qualifier
```

Computations and outputs require access to system variables. The form and number of the qualifier depend on the corresponding component.

- All qualifiers are case sensitive and must use capital letters (that is \texttt{R.V} not \texttt{R.v}).

Predefined Variables

The simulator uses predefined variables for internal computation. If these variables are used in a model description, an error message appears. All predefined variables are case insensitive.

- Predefined variables cannot be used for names in a model description.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Unit</th>
<th>Description</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>π</td>
<td>3.141592654</td>
<td>-</td>
<td>PI</td>
<td>MATH_PI</td>
</tr>
<tr>
<td>e</td>
<td>2.718281828</td>
<td>-</td>
<td>Euler number</td>
<td>MATH_E</td>
</tr>
<tr>
<td>(e_0)</td>
<td>8.85419 (10^{-12})</td>
<td>C • Jm</td>
<td>Permittivity of vacuum</td>
<td>PHYS_E0</td>
</tr>
<tr>
<td>(\mu_0)</td>
<td>1.25664 (10^{-6})</td>
<td>T • m²/J</td>
<td>Permeability of vacuum</td>
<td>PHYS_MU0</td>
</tr>
<tr>
<td>(k_B)</td>
<td>1.38066 (10^{-23})</td>
<td>J/K</td>
<td>Boltzmann constant</td>
<td>PHYS_K</td>
</tr>
<tr>
<td>e</td>
<td>1.60217733 (10^{-19})</td>
<td>C</td>
<td>Elementary charge</td>
<td>PHYS_Q</td>
</tr>
</tbody>
</table>

Predefined Constants

The simulator provides natural and mathematical constants that can be used in mathematical expressions within component dialogs or SML descriptions. The following table shows the available constants and their corresponding symbols:
Equations, Expressions, and Variables

Equations consist of operands and operators. An operand can be any number or variable name. An operator compares or assigns a value. Variables can be created and used in expressions as needed. A variable is defined when the variable name is used in an expression or for a parameter value within a component dialog. A variable does not need to be defined in a specific assignment unless it must have a defined initial value.

\[ Z := Y + X; \]

\( X, Y, \) and \( Z \) are the operands, and \( := \) and \( + \) are the operators.

If the operands are complex numbers (for example in an AC simulation), the comparison operators (<, >, <=, >=) consider only the real part.

---

### Symbol Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Unit</th>
<th>Description</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>m/s</td>
<td>Speed of light</td>
<td>PHYS_C</td>
</tr>
<tr>
<td>g</td>
<td>m/s²</td>
<td>Acceleration due to gravity</td>
<td>PHYS_G</td>
</tr>
<tr>
<td>h</td>
<td>Js</td>
<td>Planck constant</td>
<td>PHYS_H</td>
</tr>
<tr>
<td>( \theta )</td>
<td>°C</td>
<td>Absolute Zero</td>
<td>PHYS_T0</td>
</tr>
</tbody>
</table>

---

### Standard Mathematical Functions

Mathematical functions consist of the function name and one or two arguments. An argument can be any number or variable name. A mathematical function applies the function, which it represents, to the argument(s).

\[ r := \text{FCT}(x, y), r := \text{FCT}(z) \]

\( x, y, \) and \( z \) are arguments, \( z \) is a complex number, \( \text{FCT} \) is the function name, \( r \) is the result.

If the argument(s) are complex numbers (for example in an AC simulation), the functions RAD,
DEG, DEGEL, MOD, INT, FRAC, LOOKUP consider only the real part.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN(x)</td>
<td>Sine, x[rad]</td>
<td>SIN(PI/6)=0.5</td>
</tr>
<tr>
<td>COS(x)</td>
<td>Cosine, x[rad]</td>
<td>COS(2*PI/3)=-0.5</td>
</tr>
<tr>
<td>TAN(x)</td>
<td>Tangent, x[rad]</td>
<td>TAN(PI/4)=1</td>
</tr>
<tr>
<td>ARCSIN(x); ASIN(x)</td>
<td>Arc sine [rad]</td>
<td>ASIN(0.5)=0.524=PI/6</td>
</tr>
<tr>
<td>ARCCOS(x); ACOS(x)</td>
<td>Arc cosine [rad]</td>
<td>ACOS(0.5)=1.0471=PI/3</td>
</tr>
<tr>
<td>ARCTAN(x); ATAN(x)</td>
<td>Arc tangent [rad]</td>
<td>ATAN(1)=0.785=PI/4</td>
</tr>
<tr>
<td>ARCTAN2(x, y)</td>
<td>Arc tangent2 [rad]</td>
<td>ATAN2(25.1)=ATAN(4)=1.325</td>
</tr>
<tr>
<td>SINH(x)</td>
<td>Sine hyperbola</td>
<td>SINH(1)=1.175</td>
</tr>
<tr>
<td>COSH(x)</td>
<td>Cosine hyperbola</td>
<td>COSH(1)=1.543</td>
</tr>
<tr>
<td>TANH(x)</td>
<td>Tangent hyperbola</td>
<td>TANH(1)=0.762</td>
</tr>
<tr>
<td>SQU(x)</td>
<td>Square</td>
<td>SQU(16)=16²=256</td>
</tr>
<tr>
<td>X^Y</td>
<td>Power</td>
<td>7^2=49</td>
</tr>
<tr>
<td>SQRT(x)</td>
<td>Square root</td>
<td>SQRT(9)=√9=3</td>
</tr>
<tr>
<td>ROOT(x, y)</td>
<td>n-th Root</td>
<td>ROOT(27,3)=3√27=3</td>
</tr>
<tr>
<td>EXP(x)</td>
<td>Exponential function</td>
<td>EXP(5)=e^5=148.41</td>
</tr>
<tr>
<td>ABS(x)</td>
<td>Absolute value</td>
<td>ABS(-8.5)=</td>
</tr>
<tr>
<td>LN(x)</td>
<td>Natural logarithm</td>
<td>LN(3)=log_e3=1.099</td>
</tr>
<tr>
<td>LOG(x, y)</td>
<td>Common logarithm</td>
<td>LOG(7,4)=log_{10}4=1.403</td>
</tr>
<tr>
<td>INTEG(x)</td>
<td>Integration of a variable from the function call until to the simulation end</td>
<td>INTEG(var1)=∫var1 dx</td>
</tr>
<tr>
<td>RE(z)</td>
<td>Real part</td>
<td>RE(z)=5</td>
</tr>
<tr>
<td>IM(z)</td>
<td>Imaginary part</td>
<td>IM(z)=3</td>
</tr>
<tr>
<td>ARG(z)</td>
<td>Argument of a complex number in radians</td>
<td>ARG(z)=0.53</td>
</tr>
<tr>
<td>SGN(x)</td>
<td>Sign dependent value (-1, 0, 1)</td>
<td>SGN(3)=1; SGN(0)=0; SGN(-3)=-1</td>
</tr>
<tr>
<td>RAD(x)</td>
<td>Conversion from degrees to radians</td>
<td>RAD(30)=PI/6=0.524</td>
</tr>
<tr>
<td>DEG(x)</td>
<td>Conversion from radians to degrees</td>
<td>DEG(PI/2)=90°</td>
</tr>
<tr>
<td>DEGEL(x, y)</td>
<td>Conversion from degrees electrical to seconds with respect to Hz</td>
<td>DEGEL(180,50)=10ms</td>
</tr>
<tr>
<td>MOD(x, y)</td>
<td>Modulus</td>
<td>MOD(370,60)=10</td>
</tr>
<tr>
<td>INT(x)</td>
<td>Integer part of a value</td>
<td>INT(2.5)=2</td>
</tr>
<tr>
<td>FRAC(x)</td>
<td>Fractional part of a value</td>
<td>FRAC(2.5)=0.5</td>
</tr>
</tbody>
</table>
Network Configurations

In SIMPLORER only ammeters are allowed as controlling components for current controlled elements. These must be inserted properly in the controlling branch. If sources are part of mutual controlling sources in the circuit, stability problems may occur if the total gain of the loop is greater than or equal to one.

The following types of network configurations are invalid:

- Series connection of ideal current sources
- Series connection of inductors and ideal current sources
- Series connection of inductors with different initial values of current, $I_0_1 \neq I_0_2$
- Series connection of an inductor with an initial current value and an opened ideal switch or nonconducting system level semiconductor
- Parallel connection of ideal voltage sources
- Parallel connection of capacitors with ideal voltage sources
- Parallel connection of capacitors with different initial values of voltage, $V_0_1 \neq V_0_2$
- Meches which consist only of ideal sources (short-circuit)
- Open-ended branches

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOKUP(x,y)</td>
<td>Access function to a characteristic</td>
<td>LOOKUP(XY1.VAL,5)= Y value of the characteristic XY1 for the X value 5</td>
</tr>
<tr>
<td>IF (condition) { var:=1; } ELSE IF (condition) { var:=2; } ELSE { var:=3; }</td>
<td>If-Else function to perform operations dependent on conditions The ELSE IF and ELSE statement can be omitted</td>
<td>IF (t&gt;=1) { var:=1; } ELSE IF (t&gt;=2) { var:=2; } ELSE { var:=3; }</td>
</tr>
<tr>
<td>DB(x)</td>
<td>Conversion to Decibel (available in the DAY Post Processor only)</td>
<td>DB(8)=20\cdot\log(8)=18.062</td>
</tr>
</tbody>
</table>

When entering these functions, do not leave any space between the function argument, e.g., MOD and the open parenthesis mark.

If arguments are defined for trigonometric functions, poles must be considered to avoid possible errors during the simulation.
Actions in States

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Syntax in Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALC</td>
<td>The variable is calculated at each simulation step and each transition from one state to another.</td>
<td>var1 := 100*t</td>
</tr>
<tr>
<td>STEP</td>
<td>The variable is calculated at each valid simulation step.</td>
<td>var2 := 2*t</td>
</tr>
<tr>
<td>CATI</td>
<td>The variable is calculated outside the state graph and before the calculation of the electric circuit.</td>
<td>var3 := sqrt(t)</td>
</tr>
<tr>
<td>SET</td>
<td>The variable is calculated only once at the moment of activation of the state.</td>
<td>var4 := 2.5</td>
</tr>
<tr>
<td>DEL</td>
<td>Sets a delay. The variable is set to false at the moment of activation and set to true after the delay time.</td>
<td>var## := #time [s] del## := #10m</td>
</tr>
<tr>
<td>DELRES</td>
<td>Deletes a defined delay variable.</td>
<td>del4</td>
</tr>
<tr>
<td>DIS</td>
<td>The variable value (and moment) is displayed in the simulator status window.</td>
<td>Name.Qualifier dc.n</td>
</tr>
<tr>
<td>TXT</td>
<td>The given text string is displayed in the simulator status window.</td>
<td>&quot;Text String&quot; &quot;State waiting&quot;</td>
</tr>
<tr>
<td>KEY</td>
<td>Sets a mark in the state graph by pressing a key.</td>
<td>&lt;A&gt;</td>
</tr>
<tr>
<td>STOP</td>
<td>Interrupts the simulation (can be continued).</td>
<td>No Parameters</td>
</tr>
<tr>
<td>BREAK</td>
<td>Finishes the simulation.</td>
<td>No Parameters</td>
</tr>
<tr>
<td>SAVE</td>
<td>Saves the active simulation status in a status file.</td>
<td>No Parameters</td>
</tr>
</tbody>
</table>

Basic Rules for Specifying Time Steps

Correct simulation processing and results depend on the proper choice of minimum and maximum values for the integration step size. The smaller the maximum integration step size, the more correct the results, but the longer the processing time.

This means that, when specifying these minimum and maximum time step values, there must be a compromise between accuracy and simulation time. The basic rule of measurement “Not as precise as possible, but as precise as required” is also valid for a simulation. The following guidelines should help with the proper choice of integration step width:

<table>
<thead>
<tr>
<th>Model properties</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the smallest time constant ($\tau_{\text{min}}$) of the electric circuit (R*C or L/R) or of the block diagram (PTn-elements)</td>
<td>$H_{\text{min}} &lt; \frac{\tau_{\text{min}}}{10}$</td>
</tr>
<tr>
<td>What is the largest time constant ($\tau_{\text{max}}$) of the electric circuit (R*C or L/R) or of the block diagram (PTn-elements)</td>
<td>$H_{\text{max}} &lt; \frac{\tau_{\text{max}}}{10}$</td>
</tr>
<tr>
<td>Which is the smallest cycle ($T_{\text{min}}$) of oscillations that can be expected (natural frequencies of the system or oscillating time functions)</td>
<td>$H_{\text{min}} &lt; \frac{T_{\text{min}}}{20}$</td>
</tr>
<tr>
<td>Which is the largest cycle ($T_{\text{max}}$) of oscillations that can be expected (natural frequencies of the system or oscillating time functions)</td>
<td>$H_{\text{max}} &lt; \frac{T_{\text{max}}}{20}$</td>
</tr>
</tbody>
</table>
Appendix

- Select the smallest of each estimated maximum and minimum time step for the simulation model.
- All recommended values are based on numeric requirements and experience and do not guarantee a successful simulation. Please consider the algorithm as a guideline.
- In case of doubt, decrease the maximum and minimum step size by dividing by 10, repeat the simulation and compare the results. If the second set of results (with the step size decreased) shows conformity with the first results, then the step sizes chosen for the first simulation were appropriate (remember that smaller values increase the simulation time).

If the number of iterations is identical with the defined maximum value (Maximum Number of Iterations, or Iteratmax) during the simulation, the model may be incorrect. The simulation monitor displays the actual number of iterations for each step of the simulation.

<table>
<thead>
<tr>
<th>Model properties</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the smallest controller sampling (TS\textsubscript{min})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H\text{min} &lt; \frac{TS\textsubscript{min}}{5}</td>
</tr>
<tr>
<td></td>
<td>H\text{max} = TS\textsubscript{min}</td>
</tr>
<tr>
<td>What is the fastest transient occurrence (TU\textsubscript{min}) (edge changes of time functions)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H\text{min} &lt; \frac{TU\textsubscript{min}}{20}</td>
</tr>
<tr>
<td>What is the time interval to be simulated (Tend)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H\text{max} &lt; \frac{TEND}{50}</td>
</tr>
</tbody>
</table>
A.4 Troubleshooting

Modeling

- Project must be copied before it can be worked with
  Please remove the write protection from the files.

- Cannot connect elements on a sheet
  SIMPLORER schematic checks that the types of two pins to be connected are correct, e.g., it is not possible to connect an electrical pin to a signal pin. So please check that the proper pin types are being connected. If the connection is still impossible, please place the same element again.

- Connections are incorrect
  Select the improperly connected wires, click the right mouse button and select «Disconnect» from the pop-up menu.

- Errors in the SML Syntax
  Using comma as decimal point is not allowed.
  Logic operators must be surrounded by spaces.
  Predefined variables cannot be used for names in a model description.
  When using functions, there should be no space between the function argument and the open parenthesis mark.
  Names for variables and elements are case sensitive. Please check the style.

- Errors in modeling
  Each independent circuit has to be connected to the ground node at least once. Only electrically reasonable circuits lead to correct simulation results. Voltage sources, current sources and switches are ideal components.

Display and Simulation

- No graphic is displayed
  It is possible that there are no outputs defined for a component (select «Output» from the Schematic sheet’s shortcut menu) or no values were selected for display in a Display Element.
  Please check the settings in EDIT>PROPERTIES. Check also, if the View Tool Window is reduced to a symbol in the Windows task bar. Restore it with the menu item «Restore».

- Simulation does not start
  Check the simulation queue in the SSC Commander to see all active jobs. All simulations are placed in this queue and processed according the start time. Another possibility is that the simulator window is still open in the task bar. If so, please close it.

- Unexpected simulation results
  If the controlled sources are part of closed control loops in the circuit, stability problems can occur if the total gain of the loop is greater than or equal to one. In this case, (linear or nonlinear) controlled sources which directly access variables from the set or circuit equations should be used.

- The simulator computes always with the maximum step width (HMAX)
  If the simulation model only consist of blocks, the block diagram is computed (to each time step) with the maximum step width (HMAX).
A.5 Literature Reference


[3] Vahe Caliskan
   Modeling and Simulation of a Claw-Pole Alternator, Detailed and Averaged Models,
   LEES Technical Report TR-00-009, Laboratory for Electromagnetic and
   Electronic Systems, Massachusetts Institute of Technology,
   Cambridge, MA, October 2000

   Electromechanical Dynamics, Part 1, 1968

   Analytical Derivation of a Coupled-Circuit Model of a Claw-Pole Alternator
   with Concentrated Stator Winding,
Index

Symbols
^ (SML function power) 221

Numerics
2D Digital Graph 23, 46
2D View 22, 29

A
ABOVE attribute 115, 119, 207
ABS (SML function absolute value) 221
ACOS (SML function arc cosine) 221
Across quantities 12
Action types 223
Add Ones tab 217
Admittance load model 91
AFTER 201
Alias for file names 209
Alias for model library name 15, 209
Alternator example
  Creating simulation model 29
  Simulation results 34
  system_alternator.ssh 29
Alternator model 31
AMS tab 217
AND (logic operator SML) 220
Animation 77, 101
Appendix 211
ARCCOS (SML function arc cosine) 221
ARCHITECTURE statement 167
Architectures 16
  multiple 132
ARCSIN (SML function arc sine) 221
ARCTAN (SML function arc tangent) 221
ARCTAN2 (SML function arc tangent 2) 221
ARG (SML argument function) 221
Arithmetic operators
  SML 220
  VHDL-AMS 196
ASIN (SML function arc sine) 221
ASSERT statement
  Concurrent 181
  Sequential 185
Assignment operators
  SML 220
  VHDL-AMS 205
ATAN (SML function arc tangent) 221
ATAN2 (SML function arc tangent 2) 221
Attribute declaration 178

Attributes 199
  For quantities 199
  For signals 200
Automotive alarm system model 118
Automotive Powernet system example 25

B
Basics tab 217
Battery example 65
  Behavioral description 69
  case_study_battery.ssh 65
  Creating simulation model 35
  Graphical description 67
  Results 70
  Simulation results 39
  Structural description 68
  system_battery.ssh 35
Battery model 36
  Behavioral description 69
  Graphical description 67
  Structural description 68
BLOCK statement 179
BREAK (Action type) 223
BREAK statement
  Concurrent 183
  Sequential 189

C
CALC (Action type) 223
Capacitor
  VHDL-AMS example 16, 17
CASE statement
  Sequential 187
  Simultaneous 192
Case studies 63
  Automotive alarm system 118
  Battery model 65
  Claw-Pole model 81
  DC-DC model with PWM 114
  Electro-mechanical subsystems 103
  Fuse model 71
  Load models 90
  Using example sheets 63
VHDL-AMS modeling features 64
CATI (Action type) 223
Claw-Pole alternator
  Averaged model 87
  Detailed model 81
Claw-Pole example 81
   Averaged model 87  
   case_study_clawpole_avg.ssh 81  
   case_study_clawpole_math.ssh 81  
   Detailed model 81  
   Results 86, 89
Color schemes (Display Element) 24
Common conventions (SML) 218
Compile function 161
COMPONENT declaration 178
Component instantiation 69, 75, 99
Component instantiation statement 182
Components
   Outputs 21
Concurrent statements 179
   ASSERT statement 181
   BLOCK statement 179
   BREAK statement 183
   Component instantiation statement 182
   Concurrent procedure call statement 180
   Concurrent signal assignment statement 181
   PROCESS statement 180
Configuration files 54
Configuration modes example
   Simulation results 55
   system_panel_config.ssh 50
Connecting models 18
Conservative Nodes tab 153
CONSTANT declaration 175
Constants
   SML 219
Control panel model 51
Conversion
   From degrees electrical to seconds (function) 221
   From degrees to radians (function) 221
   From radians to degrees (function) 221
Copy subsheets 134
COS (SML function cosine) 221
COSH (SML function cosine hyperbola) 221
Data object declarations 175
   CONSTANT declaration 175
   FILE declaration 176
   SIGNAL declaration 176
   VARIABLE declaration 176
Data types 18, 198
DB (function) 222
DC-DC converter example
   Creating simulation model 40
   Simulation results 44
   system_dcdc.ssh 40
   DC-DC converter model 42
DC-DC model with PWM 114
   case_study_pwm_dcdc.ssh 114
   PWM compiler 115
   Results 117
Declerations 174
Defining
   Model properties 19
   Outputs 21
DEG (SML function) 221
DEGEL (SML function) 221
DEL (Action type) 223
Delay variable 223
DELTRES (Action type) 223
Design units 166
Digital circuit modeling 114
Digital controller model 122
Digital tab 217
DIS (Action type) 223
Display Elements 21
   2D Digital Graph 23, 46
   2D View 22, 29
Displays tab 217
DOT attribute 17, 133, 142, 148
Drive system 103
Duplicating subsheets 134
Electro-mechanical subsystems 103
   case_study_em_linear_drive.ssh 103
   case_study_em_solenoid.ssh 103
   Linear drive system 103
   Results 107, 113
   Solenoid system 108
   ELEMENT ModelAgent 138
   Encoding model text 155
   Entities 16
ENTITY statement 167
Equations (SML) 220
Examples 3
EXIT statement 188
EXP (SML exponential function) 221
Expressions
   SML 220
   VHDL-AMS 196
FILE declaration 176
File formats 152
FILE ModelAgent 136
FRAC (SML function fractional part of a value) 221
Functions 32, 33, 173
Functions (SML) 220
Fuse example 71
Fuse model example 71

G
Gearbox model 106
General properties tab 151
General tab 160
Glossary 211
Graphic card 3
Graphical subsheets 132

H
HMAX 223
HMIN 223

I
Identifiers 195
IF (SML function If-Else) 222
IF statement 186
Sequential 186
Simultaneous 191
Ignition switch and loads example 47
Defining simulation model 45
Simulation results 49
system_ignition_loads.ssh 45
Ignition switch model 47
IM (SML imaginary part function) 221
Importing VHDL-AMS models 77
Installation 3
Operating systems 3
Web-Update 10
Instantiation 69, 75, 99
INT (SML function integer part of a value) 221
INTEG (SML integration function) 221
INTEG attribute 44
Introduction 1

K
KEY (Action type) 223
Keywords 195
Model 152
SML 219
VHDL-AMS 201

L
Lamp model 98

Language support 139, 152
Linear drive system 103
Literals 195
LN (SML function natural logarithm) 221
Load models 90
Admittance load 91
case_study_loads.ssh 90
Nominal load 94
Results 93, 96, 98, 100
Switched load 96
Load reference arrow system 12
LOG (SML function common logarithm) 221
Logical operators 12
SML 220
VHDL-AMS 196
LOOKUP (SML access function to a characteristic) 222
LOOP statement 187

M
Manuals 3
Manufacturers tab 217
Mathematical functions (SML) 220
Menu sequence 2
Mixed-Signal Models 206
MOD (SML function modulus) 221
Model development 131
Model libraries 135, 136
Adding models 150
Alias for library name 15, 209
Compacting and repairing 137
Managing 136
Packages and models 14
Model properties 151
Model symbols 141
Modifying 149
ModelAgent 9
Application 135
Library and model management 135
Symbols 140
Working window 135
Modeling 12
Aspects in SIMPLORER 205
Multidomain 71, 103
Troubleshooting 225
Modeling style 3
Behavioral 65, 69
Graphical 65, 67
Structural 65, 68
Index

Models 12, 136
    Copying 12
    Copying in the ModelAgent 138
    Creating VHDL-AMS models 142
    Defining properties 19
    Deleting in the ModelAgent 138
    Exporting 150
    File formats 152
    For VHDL-AMS simulation 162
    Importing 77
    In model libraries 14
    Managing 138
    Mixed-signal 206
    Modifying symbols 149
    Moving in the ModelAgent 138
    Outputs 21
    Placing and connecting 18
    VHDL-AMS, creating in the ModelAgent 142, 143
    VHDL-AMS, creating with the wizard 71, 146
Motor model 104
Multidomain modeling 71, 103
Multidomain modeling packages 171

N
Names
    SML 218
    VHDL-AMS 195
NATURE declaration 175
Natures 12
Network configurations 222
NEXT statement 188
Nominal load model 94
Non-Conservative Nodes tab 153
NOT (logic operator SML) 220
NULL statement
    Sequential 189
    Simultaneous 194
Numeric data 218

O
Object browser 9
OmniCustoms 18, 90, 118
Online help 3
Operating systems 3
Operators
    SML 220
    VHDL-AMS 196
    OR (logic operator SML) 220
Outputs 21
    2D Digital Graph 23
    In Display Elements 21
    In Model Dialogs 21

P
Package
    Body 168
    Declaration 168
    Visibility 169
Package Body Text tab 161
Package properties 160
Package Text tab 160
Packages 158, 168
    Creating in ModelAgent 158
    For multidomain modeling 171
    In model libraries 14
    Inserting in ModelAgent 160
    Standard 169
Parameter names 19
Parameter sets 156
Parameters
    Defining tolerance for 155
    Of models 19
    Simulator 223
    Stress analysis 157
    VHDL-AMS Simulator 163
Pentium 3
Placing models 18
Powernet example 25
    Alternator (Step 1) 29
    Alternator model 31
    Battery (Step 2) 35
    Battery model 36
    Configuration modes (Step 5) 50
    Control panel model 51
    DC-DC converter (Step 3) 40
    DC-DC converter model 42
    Ignition switch and loads (Step 4) 45
    Ignition switch model 47
    Model overview 26
    Powertrain (Step 6) 56
    Using example sheets 27
    VHDL-AMS modeling features 28
Powertrain example
    Creating simulation model 56
    Simulation results 61
    system_powertrain.ssh 56
Probe 21, 67, 154
Procedure call statement
    Concurrent 180
    Sequential 186
Procedures 172
PROCESS statement 180
Processes 205
Project tab 217
Projects
    Creating 7
    Titles 7
PWM controller 115
SIMPLORER — VHDL-AMS Tutorial

Q
Qualifier of parameters 219
Quantities 205
Quantity attributes 199

R
RAD (SML function) 221
RAMP attribute 60, 91, 94, 116, 206
RE (SML real part function) 221
Reference system 12
Relational operators
SML 220
VHDL-AMS 196
Report browser 9
Reserved words (VHDL-AMS) 201
Results 21
RETURN statement 189
ROOT (SML function root) 221

S
SAVE (Action type) 223
Schematic
Starting 6
Working window 9
Sensor modeling 118
Sequential statements 184
ASSERT statement 185
BREAK statement 189
CASE statement 187
EXIT statement 188
IF statement 186
LOOP statement 187
NEXT statement 188
NULL statement 189
Procedure call statement 186
RETURN statement 189
Signal assignment statement 185
Variable assignment statement 186
WAIT statement 184
SET (Action type) 223
Setup 4
SGN (SML function sign dependent value) 221
Shift operators 196
Shortcut menu 2
SI Units 218
Signal assignment 206
Signal assignment statement
Concurrent 181
Sequential 185
Signal attributes 200
SIGNAL declaration 176
Signals 205
Simple simultaneous statement 191
SIMPLORER
Creating a new project 6
Documentation 3
Installation 3
Language support 139, 152
ModelAgent application 135
Online help 3
Printed manuals 3
Simulation Center (SSC) 1
Starting simulation system 5
Version sizes 4
VHDL-AMS models 11
Web-Udates 10
Simulation
Examples 3
Outputs 21
Parameters 163
Results 21
Simulation tab 153
Simulator
Environment 162
Parameters 163
Solvability 208
Simultaneous procedural statement
Statement 193
Simultaneous statements 191
CASE statement 192
IF statement 191
NULL statement 194
Simple simultaneous statement 191
Simultaneous procedural statement 193
SIN (SML function sine) 221
SINH (SML function sine hyperbola) 221
SML keywords 219
Solenoid model 109
Solenoid system 108
Solvability 208
Speed-Voltage model 87
SQRT (SML function square root) 221
SQU (SML square function) 221
Standard mathematical functions (SML) 220
Starting
Schematic 6
Setup 4
SIMPLORER 5
Statements
Concurrent 179
Sequential 184
Simultaneous 191
Stator-Impedance model 88
STD STANDARD 169
STEP (Action type) 222
Stimulus generator model 120
STOP (Action type) 223
Stress analysis parameters 157
Index

Subprograms 172
  Functions 32, 33, 173
  Procedures 172
Subsheets 132, 132
  Adding to a library 134
  Copying 134
  Duplicating 134
  Graphical 132

SUBTYPE declaration 175
Suffixes (SML) 218
Switched load model 96
Symbol
  animation 77
  Symbol animation 101
  Symbol Editor 101, 149
  Symbols
    ModelAgent 140
    System constants 219

T
TAN (SML function tangent) 221
TANH (SML function tangent hyperbola) 221
Text in menus 2
Text subsheets in VHDL-AMS 133
The 207
Through quantities 12
Time step 223
Tolerance for parameters 155
Tools tab 217
Transformation components 18
Troubleshooting 225
  Display and Simulation 225
  Modeling 225
Tutorial
  Content 1
TXT (Action type) 223
TYPE declaration 174
Type declarations 174
  NATURE 175
  SUBTYPE 175
  TYPE 174

U
User names 5
Users tab 217

V
Values on sheet 210
Variable assignment statement 186
VARIABLE declaration 176
Variables
  SML 219, 220
  VHDL-AMS 205
Vector inputs on sheet 210
Version size 4
VHDL-AMS
  Capacitor (nonlinear) 17
  Capacitor (static) 16
  Case studies 63
  Concurrent statements 179
  Definitions (packages) 140
  Design units 166
  Entities and Architectures 16
  Exporting library 137
  Exporting model 138
  Language history 11
  Model wizard 146
  Models in SIMPLORDER 11
  Sequential statements 184
  Simulator 162
  Simulator parameters 163
  Simultaneous statements 191
  Standard Packages and Types 169
  Subsheets 132
  Subsheets in Schematic 132
  Syntax checking 133
  Using models 12
  Wizard 71

W
WAIT statement 184
Web-Update 10
Windows 2000 3
Windows XP 3
WORK library 208